UNIVERSIDAD CARLOS III DE MADRID

Escuela Politécnica Superior - Leganés

INGENIERÍA DE TELECOMUNICACIÓN

Proyecto Fin de Carrera

Herramienta de Optimización Paramétrica Distribuida con HFSS

AUTOR: Carlos Mariano Lentisco Sánchez.
DIRECTOR: Ignacio Martínez Fernández.
TUTOR: Luis Emilio García Castillo.

Leganés, 2013

Título: *Herramienta de Optimización Paramétrica Distribuida con HFSS.*

Autor: *Carlos Mariano Lentisco Sánchez*

Director: *Ignacio Martínez Fernández*

Tutor: *Luis Emilio García Castillo*

La defensa del presente Proyecto Fin de Carrera se realiza el día xx de xxxxx de 2012; siendo calificada por el siguiente tribunal:

Presidente:

Secretario

Vocal

Habiendo obtenido la siguiente calificación:

Calificación:

Presidente	**Secretario**	**Vocal**

Agradecimientos

En primer lugar, quiero agradecer al Grupo de Radiofrecuencia del departamento de Teoría de la Señal y las Comunicaciones la oportunidad que se me ha brindado. En especial, a Nacho por su trabajo, esfuerzo, dedicación y por la confianza que ha depositado en mí, y a Luis Emilio, por darme la posibilidad de realizar este proyecto.

A mi familia, porque gracias a ellos he crecido en el amor y en la felicidad. En primer lugar, a mi padre, que está ya con Dios. A mi madre, por su infinito amor y sacrificio, porque ser como es, me ha hecho ser como soy. A mi hermano, porque su apoyo siempre ha sido pleno y determinante para mí, por su ejemplo y por ser un amigo y un hermano a la vez.

A mi novia Marijose, por este año de amor sin medida y por hacerme aún más feliz. Por compartir los momentos más duros de mi etapa universitaria con una sonrisa y mucha paciencia. Por regalarme un futuro, que junto a ti, es ilusionante.

A mis amigos de siempre, por los recuerdos, por todos los momentos compartidos: las fiestas, las risas, las penas, los conciertos, ... Gracias a todos y cada uno de vosotros, gracias por acompañarme en esta aventura.

A mis amigos de la universidad, los **ÚNICOS** que saben lo que es esta carrera en esta universidad: Gracias Yolanda, Armada, Ainhoa, Isabel, Baltasar, David, Jonathan, Sandra, Alberto, Dani y tantos y tantos otros..., y en especial, como no podía ser de otra forma, a Álvaro por su gran amistad, porque amigos para siempre means you'll always be my friend.

Y como no, al rock'n'roll consumido durante estos años

Gracias

You can have it all but, how much do you want it?.
Noel Gallagher

Resumen

Hoy en día, las señales de naturaleza electromagnética son la esencia de las actuales tecnologías de la información y de las comunicaciones. En particular, el auge de las redes de comunicación inalámbrica, ha elevado la demanda de diseño de estructuras electromagnéticas de gran complejidad.

El análisis de estas estructuras es costoso en tiempo y en recursos computacionales, por lo que, realizar simulaciones paramétricas, que requieren realizar múltiples modificaciones sobre los parámetros de la estructura, eleva el conjunto de análisis y aumenta inevitablemente el tiempo de ejecución.

El objeto del presente Proyecto Fin de Carrera es diseñar una metodología de simulación paramétrica distribuida que permita gestionar de forma eficiente los recursos computacionales que se disponen, minimizando así, dicho tiempo de ejecución. A esta metodología se le puede incluir, si es el caso, un optimizador que ayude al diseñador a encontrar la solución óptima. En este proyecto se propone un optimizador que hace uso de la simulación paramétrica para encontrar los valores de los parámetros que proporcionan la solución óptima.

Se ha utilizado HFSS como simulador de problemas electromagnéticos, tanto para el modelado de las estructuras, como también para su análisis y obtención de resultados. El conjunto de análisis a realizar en el proceso de optimización es distribuido en el cluster del Grupo de Radiofrecuencia del departamento de Teoría de la Señal y Comunicaciones, gracias al sistema gestor de colas Sun Grid Engine (SGE).

La herramienta ha sido validada con la optimización de un filtro de guiaonda con postes dieléctricos, en particular, de cuatro cavidades y cinco irises. Como puede intuirse, esta estructura cuenta con un elevado número de parámetros, lo que justifica una optimización con un uso eficiente de los recursos debido a la elevada necesidad de cómputo necesaria.

Abstract

Nowadays, electromagnetic natural signals are the essence of modern information technology and communications. Particularly, the growth of wireless communication networks has risen the demand of highly complex electromagnetism structures design.

The analysis of these structures is costly in time and computational resources. Therefore, carrying out parametric simulations, which require making several modifications over the structure parameters, expands the analysis set and increases inevitably the execution time.

The object of the following dissertation is to design a methodology of distributed parametric simulation that allows to efficiently manage the available computational resources, minimizing this way the execution time. To this methodology can be included, if needed, an optimizer which helps the designer to find the optimal solution. In this project it is proposed an optimizer using the parametric simulation to find the values of the parameters which give the optimal solution.

HFSS has been used as the electromagnetic problem simulator, both for structure modelling and for analysis and obtaining of results. The set of analysis to make in the optimization process is distributed in the cluster from the Radio Frequency Group of the Signal Processing and Communication Theory Department, thanks to the Sun Grid Engine (SGE) queuing system.

The tool has been validated with the optimization of a waveguide filter with dielectrics, particularly using four cavities and five iris. As can be seen, this structure includes a high number of parameters, which justifies an optimization that efficiently manages resources, due to the high computation capacity needed.

Índice general

Índice de figuras

Índice de cuadros

Índice de ficheros

Capítulo 1

Introducción

Hoy en día, el auge de las redes de comunicación inalámbrica, ha elevado la demanda de diseño de estructuras electromagnéticas de gran complejidad. La resolución analítica de estas estructuras entraña una extremada dificultad, siendo este problema, a efectos prácticos, irresoluble. Por esta razón, se requiere el uso de simuladores electromagnéticos que utilicen métodos numéricos para hallar la solución. Solución que a pesar de no ser exacta, se aproxima a esta con elevada precisión.

En el proceso productivo de un dispositivo con estas características, resulta conveniente evaluar el tiempo y coste en el que se incurre para obtener el producto final. Ambos indicadores, dependen del número de prototipos experimentales que se construyen para verificar el funcionamiento del diseño. Gracias al uso de herramientas software, es posible reducir el número de prototipos y ajustes experimentales, minimizando así, el coste y tiempo requeridos en el diseño del dispositivo.

La complejidad estructural del circuito, como la inclusión de materiales dieléctricos, o el tamaño eléctrico del mismo, pueden incrementar el número de parámetros de nuestro problema de manera notable. A raíz de esto, existe la necesidad de realizar análisis de sensibilidad paramétrica para observar su influencia en la respuesta de la estructura. Las áreas de uso incluyen optimización, control óptimo, análisis de la incertidumbre, valoración del parámetro, simplificación del modelo y diseño experimental para una amplia gama de problemas de ingeniería; y en particular, de los problemas de naturaleza electromagnética, de especial interés para el Grupo de Radiofrecuencia del departamento de Teoría de la Señal y las Comunicaciones.

El análisis paramétrico consiste en realizar múltiples variaciones sobre los parámetros de la estructura, analizando su respuesta para cada combinanción paramétrica de forma independiente. De esto se deduce un incremento considerable del número de simulaciones a realizar, y por consiguiente, un incremento de la carga computacional y el tiempo de ejecución, puesto es sabido que cada análisis presenta, por sí mismo, una elevada complejidad.

El presente Proyecto Fin de Carrera hace uso de HFSS (High Frequency Structure Simulator) como simulador electromagnético, el cual usa el Método de Elementos Finitos [1] (MEF) como método numérico para aproximar la solución. Las razones de su elección se fundamentan en que el Grupo de Radiofrecuencia cuenta con diversas licencias, no obstante, el uso de otro simulador es posible para desarrollar la herramienta. Las licencias que se disponen son de caracter académico y carecen de la funcionalidad DSO (Distributed Solver Option) que ofrece ANSYS como solución para el análisis paramétrico. Es objeto de este proyecto, salvar esta limitación. Además, se ha mejorado la solución de ANSYS maximizando la eficiencia de los recursos computacionales y permitiendo un análisis paramétrico sin limitaciones, de forma que, el usuario disponga libertad de decisión en cuanto a selección y gestión de este análisis.

Con este fin, se ha desarrollado una metodología de simulación paramétrica distribuida que permite aprovechar los recursos computacionales que dispone el Grupo de Radiofrecuencia. Este grupo, cuenta con un cluster de cómputo científico [2] que se aprovechará en toda su dimensión.

El cluster es un conjunto de ordenadores que operan como si fueran uno solo. Se asemeja a la idea del principio: divide y vencerás, pero aplicándolo sobre los procesos que están ejecutándose en el ordenador. Este cluster [3] es de alto rendimiento (High Performance Computing Cluster), pero gracias al uso de un sistema de gestión de colas podemos utilizarlo como uno de alta carga (High Througput Computing Cluster). La distinción es sutil: el primero es capaz de ejecutar trabajos más grandes que aquellos que podría ejecutar cualquiera de los nodos que lo componen; mientras que el segundo tipo, está diseñado para ejecutar el máximo número de trabajos, lo que es necesario en este proyecto.

El sistema gestor de colas que se ha utilizado para que el cluster opere en un entorno HTC (High Througput Computing Cluster) es Sun Grid Engine (conocido antes como CODINE (*COmputing in DIstributed Networked Environments*), y ahora llamado OGE (*Oracle Grid Engine*). Se trata de un sistema de colas de código abierto que se encarga de gestionar y distribuir los recursos computacionales disponibles, como pueden ser los núcleos de los procesadores o memoria. Cabe resaltar que se ha utilizado SGE, y que éste puede ser sustituido por otros sistemas, como por ejemplo, CONDOR [4].

Gracias al empleo de SGE, se puede hacer un uso eficiente de los recursos computacionales del cluster, lo que nos permitirá minimizar el tiempo de ejecución drásticamente. Este hecho, aporta valor a una simulación paramétrica, que de otro modo, y dadas las características, podría llegar a ser inabordable.

Los resultados obtenidos por la simulación paramétrica distribuida son usados, como se vió anteriormente, en diversas disciplinas. En el presente Proyecto Fin de Carrera se ha desarrollado una herramienta de optimización que sirve al usuario para obtener los valores de los parámetros que maximizan una determinada utilidad. La herramienta hace uso de un algoritmo de búsqueda para encontrar la solución óptima, en este particular y por las justificaciones que se recogen en este documento (capítulo 4), se ha seleccionado un algoritmo genético.

Para mejorar la experiencia del usuario se ha automatizado un conjunto de tareas que permiten el manejo de la herramienta con algunos conceptos básicos relativos a la inicialización del sistema y al sistema gestor de colas. De esta forma, se evita la interacción con el sistema en el proceso de optimización.

La arquitectura propuesta da la posibilidad de que un usuario pueda utilizar la herramienta en remoto, para ello, solo tiene que acceder con su ordenador personal y, a través de internet, al cluster del Grupo de Radiofrecuencia. El acceso se realiza a la red del departamento de Teoría de la Señal y las Comunicaciones, y dentro de la red, al nodo maestro. De esta forma, se tiene acceso a cualquier máquina del cluster. Este cluster está basado en Rocks y opera sobre un entorno Linux. El usuario, sin embargo, puede operar sobre un entorno Windows, siempre y cuando, disponga de las herramientas software necesarias para poder operar Linux desde un entorno Windows, por ejemplo, NX Client o Putty y Xming.

En este capítulo se ha definido la motivación y descrito el entorno de trabajo. En el siguiente capítulo se describe HFSS y SGE con mayor profundidad (capítulo 2), mostrando la funcionalidad que soporta cada una de estas herramientas y las diversas opciones que ofrecen.

Tras las adquisición de los conocimientos necesarios sobre las herramientas empleadas, se procede a describir la metodología usada para realizar los análisis que se requieren en el proceso de optimización (capítulo 3). Esta metodología está orientada a la gestión eficiente de los recursos y a un análisis paramétrico sin limitaciones. En este capítulo, se describen con detalle los scripts que han sido necesarios para la consecución de este objetivo.

La herramienta de optimización (capítulo 4), que hace uso de la metodología anterior, exige una explicación conveniente sobre las razones que han motivado la selección de la solución desarrollada con respecto a otras posibles soluciones. En

este capítulo se profundizará en la descripción del algoritmo encargado de dar la solución óptima y se validará dicho algoritmo mediante un ejemplo sencillo.

Una vez descrito el sistema, se procede a optimizar una estructura. En el capítulo 5, se describe la optimización de un filtro dieléctrico, detallándose como se ha configurado dicha optimización y presentando la solución obtenida, debidamente analizada.

El capítulo 6 describe cómo se ha desarrollado este Proyecto Fin de Carrera explicándose las tareas realizadas en cada fase. Para finalizar, se establecen las conclusiones y las líneas futuras (capítulo 7) mostrándose en el capítulo 8 el presupuesto que se deduce del presente Proyecto Fin de Carrera.

Capítulo 2

Herramientas software

Para la consecución del objetivo marcado en el capítulo 1, se introducen dos herramientas software de especial relevancia: HFSS y SGE; así como el escenario (red y cluster) donde se realizará el desarrollo del presente Proyecto Fin de Carrera.

Es importante resaltar que no son las únicas herramientas empleadas, ya que también es fundamental el uso de Matlab para realizar, como se verá en los siguientes capítulos:

- El algoritmo de búsqueda en el que se basa la herramienta de optimización: Algoritmo Genético (ver capítulo 4).
- Generación de ficheros VBScript en la fase *preproceso* de la simulación paramétrica distribuida (ver capítulo3).
- Post procesado de los resultados obtenidos en la fase *resolución* de la simulación paramétrica distribuida (ver capítulo 3).
- Presentación de los resultados.

Además, se han utilizado otras herramientas para la ejecución remota del sistema (capítulo 1), como pueden ser Putty, Xming o NX Client.

2.1. HFSS

Como se introduce en el capítulo 1, HFSS se utiliza para el diseño y análisis de estructuras electromagnéticas. En la simulación de un problema electromagnético se pueden distinguir las siguientes etapas:

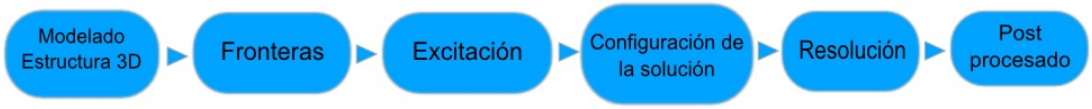


Figura 2.1: Diagrama de estados en la simulación de una estructura con HFSS.

Como se puede ver en la Figura 2.1, lo primero a realizar es el modelo de la estructura que se desea analizar, asignando sus materiales y dimensiones apropiadamente. Una vez modelada la estructura, se debe indicar la condición de frontera, HFSS supone que el material de fondo o por defecto es PEC. A continuación, se establece la fuente de campo electromagnético, voltaje o corriente que excitan la estructura. Para finalizar, y tras configurarse el setup de análisis, se resuelve la estructura y se realiza el post-procesado, que consiste en obtener los resultados y reports que el usuario desea.

2.1.1. Modelado y simulación de estructuras en HFSS

Para explicar el procedimiento descrito anteriormente, se va a detallar cómo se realiza una simulación de un dispositivo arbitrario en HFSS, desde el modelado del propio dispositivo hasta la presentación de los resultados.

En la Figura 2.2 podemos observar un ejemplo de estructura modelada con HFSS, en este caso una antena helicoidal. La interfaz cuenta con los siguientes menús y ventanas:

- Menu bar: lista todas las herramientas que dispone HFSS.
- Toolbars: acceso directo a herramientas fundamentales para la creación y simulación de dispositivos.
- Project manager: en esta ventana se muestran cada una de las operaciones que realiza el usuario sobre el modelo, están ordenadas jerárquicamente por categorías: Model, Boundaries, Excitations, Mesh operations ,Analysis ,Optimetric, Results, Port Field Display, Field Overlays, y Radiation.

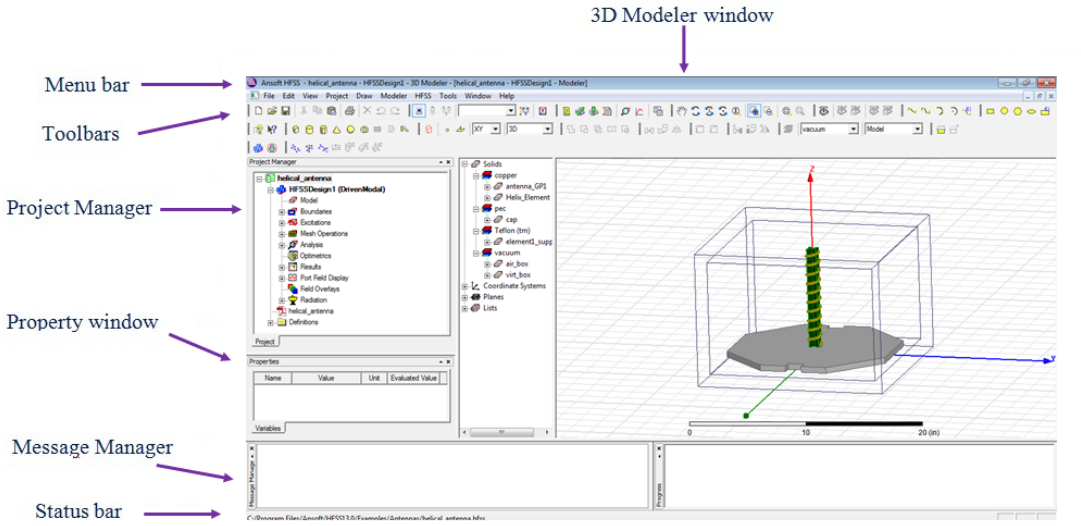


Figura 2.2: Desktop de HFSS.

- 3D modeler: en esta ventana se modela la estructura deseada. Se encuentra dividida en dos partes: a la derecha se encuentra el modelo, y a la izquierda una lista detallada y desplegable de cada una de las partes que la componen.
- Message manager: muestra una serie de avisos, advertencias y errores.
- Progress windows: muestra el estado de la simulación en curso.
- Property: indica las características del elemento seleccionado.

Una vez que se conoce y domina la interfaz, se puede describir el proceso de modelado de la estructura. En este proceso, el usuario deberá especificar algunos aspectos, como por ejemplo, dimensiones físicas y materiales que la componen.

Figura 2.3: Iconos para el modelado de la estructura.

Para dibujar el modelo se pueden utilizar las herramientas localizadas en *Toolbars* Figura 2.3. Entre las diversas posibilidades se permite dibujar curvas, polígonos y cuerpos geométricos.

Las estructuras modeladas pueden ser modificadas utilizando las herramientas señaladas en la Figura 2.4, que permiten: unir, sustraer, intersectar, dividir,

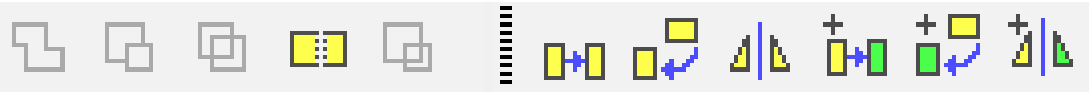


Figura 2.4: Iconos para la modificación de la estructura.

rotar y reflejar los bloques creados. Además, es posible duplicar de manera axial o rotacional el modelo.

Para el modelado y modificación de las estructuras creadas resultan útiles las herramientas indicadas en la Figura 2.5, que permiten: mover, rotar, acercar, alejar, activar y bloquear la estructura.

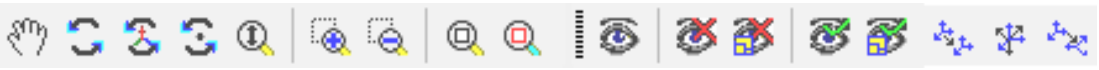


Figura 2.5: Iconos para visualización de la estructura.

A pesar de que es posible modelar la esctrura en la interfaz, HFSS permite asignar directamente la posición de inicio y las medidas físicas en la barra que se activa en la parte inferior derecha de la ventana 3D modeler, vista en la Figura 2.2.

Cuando se crea un objeto, se agrega a una lista localizada en la parte izquierda de la ventana 3D modeler. A través de ese desplegable se puede acceder por ejemplo, a las propiedades de comando. Dicha tabla describe las medidas físicas del elemento seleccionado que compone la estructura.

Finalizado el modelado de la estructura, se debe caracterizar la misma de forma que permita su análisis. Esto consiste en seleccionar alguna de las superficies que la componen e incluir excitaciones y campos electromagnéticos.

Para simular el modelo diseñado debemos establecer la configuración de análisis o *análisis setup*, el usuario puede localizarlo a través del menú principal o en el *Project Manager* (Figura 2.2).

En el *análisis setup* (Figura 2.6) se disponen varios campos para establecer el identificador del análisis o para señalar la frecuencia a la cual se ha diseñado el dispositivo. En la sección *Adaptative solution* existe la posibilidad de ajustar la cantidad de pasos máximos que se iterarán en el análisis y el máximo error permitido en busca de una malla adecuada para el modelo y frecuencia utilizada.

HFSS permite realizar un barrido de frecuencia seleccionando la opción *Add frecuency sweep*. Existen varios tipos de barrido: Discret, Interpolation, o Fast.

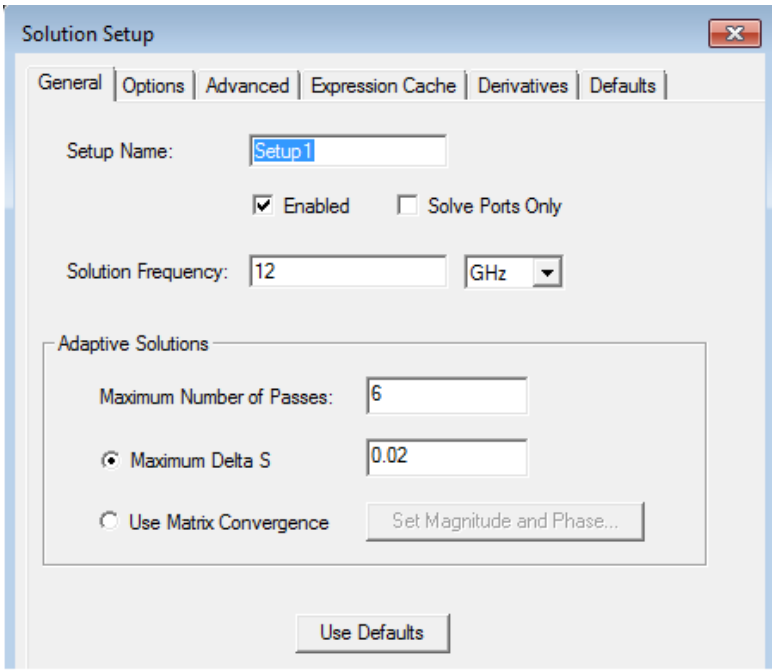


Figura 2.6: Análisis Setup.

- Discret: considera únicamente la evaluación de las frecuencias indicadas.
- Interpolation: evalúa las frecuencias indicadas y realiza una interpolación entre los valores no evaluados.
- Fast: realiza la prueba de puntos críticos y luego procede a interpolar.

Además, dispone de una herramienta de optimización. Esta, realiza análisis que permiten variar parámetros físicos del diseño y encontrar soluciones óptimas de los mismos. Dicha opción se accede a través de *Optimetric* en la ventana *Project Manager*. A pesar de disponer de este optimizador, se ha desarrollado uno propio que suple las limitaciones que plantea, por ejemplo, la finidad de parámetros a optimizar.

Una vez configurado el análisis se procede a la simulación. Para ello, se puede utilizar el menú principal o las herramientas localizadas en *Toolbars*. Estas herramientas son las mostradas en la Figura 2.7 y sirven para listar las características de diseño, comprobar la consistencia de las propiedades asignadas, iniciar la simulación o añadir notas de usuario.

Finalizado el proceso de análisis de la esctructura, se presentan los resultados. La selección de estos se realiza como siempre a través del menú principal o en la

Figura 2.7: Herramientas de simulación.

ventana *Project Manager*, sección *Results*. Entre las diversas opciones disponibles y para el ejemplo dado (antena helicoidal), se presenta el diagrama de radiación con la ganancia total de la antena. (Figura 2.8).

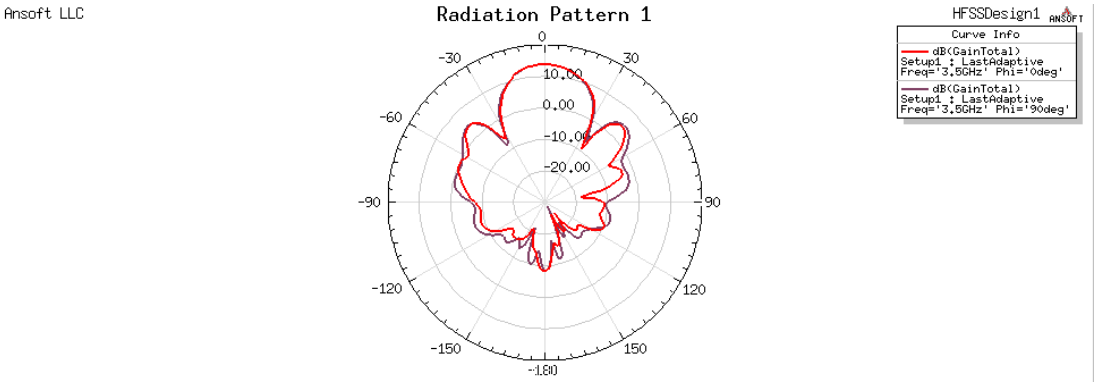


Figura 2.8: Diagrama de radiación de antena helicoidal.

El uso de la interfaz es intuitivo y permite obtener conocimiento amplio sobre la funcionalidad que ofrece HFSS. El manual de ANSOFT [5] detalla lo aquí explicado y además profundiza sobre otras opciones soportadas, opciones no consideradas en este documento, ya que no es objeto de este estudio.

HFSS permite no solo su ejecución a través de la interfaz aquí descrita, sino que dispone la opción de ejecutarse por línea de comandos. A través de la línea de comandos se puede:

- Realizar análisis sin la interfaz gráfica sobre proyectos ya definidos y con el *análisis setup* configurado previamente.
- Ejecutar scripts desarrollados en lenguaje VBScript a través de los comandos descritos en la siguiente sub-sección.

HFSS dispone de una potente herramienta de scripting que permite al usuario realizar cualquier acción disponible en la interfaz gráfica, con una salvedad: la ejecución por medio de scripting requiere activa la interfaz gráfica, característica propia de este lenguaje de programación.

El usuario puede realizar con VBScript toda la funcionalidad descrita en esta sección, modelando un dispositivo desde su origen, caracterizándolo físicamente para su análisis electromagnético y, configurando el análisis para su posterior simulación y captura de resultados.

2.1.2. Scripting en HFSS

El usuario puede escribir un script con cualquier editor de texto o puede hacer uso de la herramienta que dispone HFSS para generar código de los diseños físicos, análisis u obtención de reports que pueden ser realizados mediante la interfaz gráfica. Este código generado, puede ser modificado a posteriori para realizar los cambios que el usuario estime convenientes.

ANSOFT proporciona un API de este lenguage de programación orientado a objetos, de forma que, el usuario dispone de herramientas necesarias para poder obtener los conocimientos que se requieren para implementar simulaciones completas.

No es objeto de este estudio el profundizar en los detalles de VBScripting en HFSS, para cualquier duda se puede consultar la documentación de ANSOFT [5].

Aun así, se presenta a continuación el siguiente ejemplo ilustrativo del lenguaje VBScript.

Ejemplo de VBScript

Este ejemplo incluye comentarios, cuyas líneas comentadas sirven al usuario para describir el código desarrollado y van precedidas del caracter '.

Fichero 2.1: Ejemplo VBScript

```
1   ' ----------------------------------------------
    ' Script Recorded by Ansoft HFSS Version 10.0
    ' 11:03 AM May 3, 2005
    ' ----------------------------------------------

    Dim oDesign
6   Dim oEditor
    Dim oModule

    Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInterface")
    Set oDesktop = oAnsoftApp.GetAppDesktop()
11  oDesktop.NewProject

    Set oProject = oDesktop.GetActiveProject
    oProject.InsertDesign "Hfss", "HFSSDesign1", "DrivenModal", ""
```

```
16 Set oDesign = oProject.SetActiveDesign("HFSSDesign1")
   Set oEditor = oDesign.SetActiveEditor("3D Modeler")

   oEditor.CreateBox Array("NAME:BoxParameters", "XPosition:=", _
   "0mm", "YPosition:=", "0mm", "ZPosition:=", "0mm", _
21 "XSize:=", "1.6mm", "YSize:=", "1.2mm", "ZSize:=", _
   "0.8mm"), Array("NAME:Attributes", "Name:=", "Box1", "Flags:=", _
   "", "Color:=", "(132 132 193)", "Transparency:=", _
   0.400000005960464, "PartCoordinateSystem:=", _
   "Global", "MaterialName:=", "vacuum", "SolveInside:=", true)
```

REM y Dim se usan para la declaración de variables. Estas pueden almacenar cualquier tipo de información. En el ejemplo anterior, las tres variables son usadas como objetos. Cuando se utiliza la herramienta de recording para la generación de código, las variables usadas como objetos empiezan siempre con el caracter 'o'.

La línea 9 del ejemplo anterior muestra la función *CreateObject* que asigna una referencia de un objeto a la variable *oAnsoftApp*, esta asignación se produce sin crearse una copia del objeto.

Con el objeto creado se pueden usar las funciones de la clase a la que pertenece, con *GetAppDesktop()* obtenemos la referencia de un objeto de otro nivel jerárquico y que permitirá el uso de las funciones propias a este nivel de jerarquía. El uso de *oDesktop.NewProject* sirve para la creación de un nuevo proyecto en HFSS.

Posteriormente, se asocia al proyecto un nuevo diseño con el nombre *HFSSDesign1* (para este caso concreto). A este diseño se asocia una edición donde puede generarse cualquier tipo de estructura, nótese, que se esta realizando la misma operación que se describía en el modelado de estructuras comentado en la subsección anterior.

En el ejemplo se ha creado una caja con las dimensiones físicas dadas como parámetros en la función *CreateBox*, donde además, se ha especificado el tipo de material que compone la estructura.

Además, pueden observarse en dicho ejemplo los distintos niveles jerárquicos que presenta HFSS en la generación de cualquier script y que pueden verse en la Figura 2.9.

Las funciones disponibles para cada nivel jerárquico se encuentran en la documentación de ANSOFT [5] y van ilustradas de ejemplos para una mejor comprensión de su uso.

A continuación, se va a describir la estructura de este lenguaje de programación, donde se van a citar los elementos básicos que se utilizan en la elaboración de cualquier script.

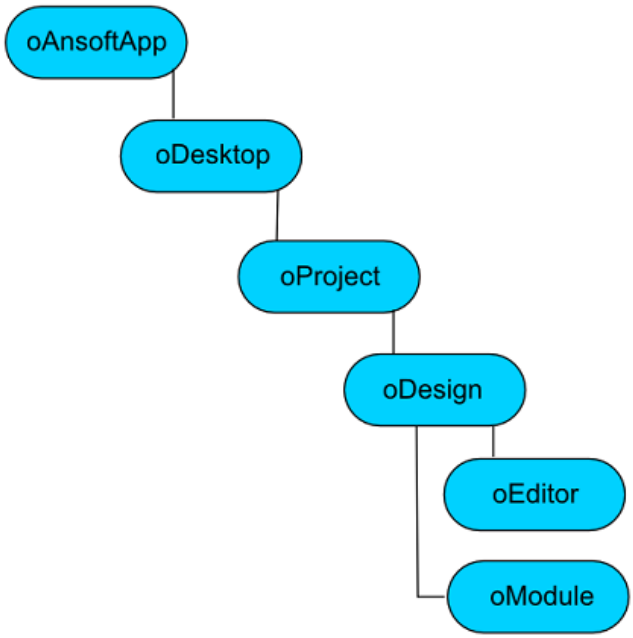


Figura 2.9: Jerarquía en la generación de un script.

Como se ha comentado anteriormente, la declaración de variables se realiza con los comandos Dim o REM. Una vez declaradas pueden almacenar cualquier tipo de información. Veamos otro ejemplo:

```
Dim box_xsize
box_xsize = "3mm"
```

También puede declararse un array de variables teniéndose en cuenta que la primera posición del array es la posición 0:

```
Dim Primitives(2)
Primitives(0) = "Box1"
Primitives(1) = "Cone1"
Primitives(2) = "Cylinder1"
```

Cuando aparecen varias operaciones en una expresión, cada parte se evalúa y se resuelve según la precedencia de los operadores. El uso de paréntesis modifica el orden de precedencia. Las operaciones entre paréntesis se realizan siempre antes, y dentro de los paréntesis, la precedencia de operadores se mantiene. Cuando las expresiones contienen operadores de más de una categoría, las operaciones aritméticas se evalúan en primer lugar, los operadores de comparación se evalúan a continuación, y los operadores lógicos se evalúan en último lugar.

Para controlar el flujo de ejecución del programa, existen unas sentencias de especial importancia: *If...Then..Else* y *Select Case*, cuya estructura se puede observar

a continuación.

Fichero 2.2: Estructura If...Then...Else

```
1
  If obj = "Box1" Then
          <statements to execute>
  ElseIf obj = "Cylinder1" Then
          <statements to execute>
6 Else
          <statements to execute>
  End If
```

Fichero 2.3: Estructura Select Case

```
  Select Case primitive_name
2 Case "Box1"
          <statements to execute>
  Case "Cylinder1"
          <statements to execute>
  Case Else
7         <statements to execute>
  End Select
```

El usuario puede implementar bucles para realizar repetidamente determinadas acciones. Esta tarea se realiza con el uso de la sentencia For...Next, cuya estructura, se muestra a continuación.

Fichero 2.4: Estructura For...Next

```
  For variable = start To end
2         <statements to execute>
  Next
```

En ciertas ocasiones el usuario tiene necesidad de convertir el tipo de datos de una variable, HFSS dispone de las siguientes funciones para esta conversión:

- CStr: convierte un entero a una variable de tipo string.
- CBool: convierte la variable pasada por parámetro a una variable de tipo booleano.
- CDbl: convierte un string a un número de doble precisión.
- CInt: convierte la variable pasada por parámetro a una variable de tipo entero.

2.1.3. Ejecución de HFSS por linea de comandos

Para ejecutar VBScripts o realizar análisis sobre estructuras definidas y con un análisis setup ya configurado, se puede recurrir a la línea de comandos, con lo que la independencia del usuario sobre la realización de una simulación puede ser completa. La ejecución de HFSS por línea de comandos se realiza con la siguiente sentencia:

hfss [options] [runCommand] [project name/script name]

Donde runCommand puede adquirir cualquiera de los siguientes valores:

- BatchSolve: analiza por defecto los setups encontrados en un proyecto HFSS. Si la solución en paralelo es posible se puede usar la función *–Distribute* en conjunción con Batchsolve. Este comando de ejecución se puede usar además con las opciones *–ng* (no gráficos) y *–WaitForLicense*. Mediante *-Remote* ‖ *-Local* ‖ *-Distributed* ejecutamos HFSS en una máquina local, remota o como análisis distribuido sobre un conjunto de máquinas.
- Batchoptions: todas las opciones accesibles en la interfaz vía Tools > Options están a nivel de usuario en el registro. Se pueden sobrescribir las entradas de dicho registro vía batchoptions. Estas opciones pueden ser leídas por línea de comandos o de un fichero, el cual debemos especificar. Esta opción es solo válida para batchjobs.
- RunScript <scriptFileName>: se ejecuta el script especificado. La opción *–ScriptArgs* se usa para pasar por parámetro argumentos al script. Se puede usar con *–Iconic*.
- RunScriptAndExit <scriptFileName>: ejecuta el script especificado y finaliza. Se puede usar la opción *–ScriptArgs* para pasar argumentos por parámetro y se puede usar con *–Iconic*, *-Logfile* y *-WaitForLicense*.

Options puede tomar cualquiera de los siguientes valores:

- -Distribute: distribuye un análisis a múltiples máquinas. Esta opción puede ser combinada con Batchsolve.
- -Iconic: ejecuta HFSS con la ventana minimizada.
- -ng: sirve para ejecutar HFSS en modo no gráfico, sin mostrar la interfaz. Es imcompatible con el uso de RunScript y RunScriptAndExit por lo que también es incompatible con la opción -ScriptArgs

- -WaitForLicense: espera hasta que una licencia queda libre, se usa en combinación con -BatchSolve or -RunScriptAndExit.
- -ScriptArgs: permite que un script reciba argumentos. Se usa en combinación con RunScript y RunScriptAndExit.

Una vez se ha realizado la llamada de ejecución de un proyecto HFSS, ya sea mediante runScript/RunScriptAndExit (se necesita la interfaz), o mediante batchsolve (no se necesita la interfaz pero deben ser proyectos con un setup ya configurado), HFSS escribe una serie de ficheros que se tendrán en cuenta posteriormente en la metodología propuesta.

Algunos de estos ficheros/directorios son los siguientes:

- nombre.hfss - Proyecto HFSS.
- design_name.hfssresults - Carpeta que contiene soluciones de campo y resultados de un diseño realizado. Se localiza donde está ubicado el proyecto HFSS.
- project_name.asol - Este fichero contiene el informe de las tareas que realiza el simulador en un análisis.

La ejecución de proyectos HFSS mediante scripts y la utilización de un sistema gestor de colas, permiten realizar una simulación distribuida (capítulo 3), para la cual se utiliza el cluster del Grupo de Radiofrecuencia.

2.2. Sun Grid Engine: SGE

Sun Grid Engine (conocido antes como CODINE (*COmputing in DIstributed Networked Environments*), y ahora llamado OGE (*Oracle Grid Engine*), es un sistema de colas de código abierto. Este sistema de gestión de colas se encarga de gestionar y distribuir los recursos computacionales disponibles, como pueden ser los núcleos de los procesadores o memoria.

Con SGE el usuario puede enviar un elevado número de tareas sin preocuparse de cómo están siendo ejecutados los trabajos o qué recursos están siendo utilizados para la ejecución de los mismos, sabiendo que el acceso a estos nunca se realizará de forma simultánea. Además, gracias a este sistema de gestión de colas,

se pueden configurar diversos parámetros como: la memoria mínima disponible, número de *cores* (unidades de procesamiento) a emplear, memoria máxima que puede utilizar la simulación, etcétera. Haciendo uso de ellos podemos optimizar el rendimiento del cluster.

Su uso resulta especialmente interesante para múltiples usuarios, donde su uso se convierte en una necesidad.

Cabe resaltar que se ha utilizado SGE, y que éste puede ser sustituido por otros sistemas de gestión de colas, como por ejemplo, CONDOR [4].

2.2.1. Uso de SGE

En esta sección se profundiza en el uso de SGE definiendo sus comandos, su funcionalidad y, mostrando ejemplos de su funcionamiento. SGE como sistema gestor de colas, cuenta con comandos para mandar tareas a la cola, monitorizar el estado de los trabajos lanzados, eliminar tareas o para configurar el sistema.

Para ejecutar trabajos en un cluster con SGE se pueden utilizar scripts, lo que resulta cómodo e intuitivo, más aún cuando no estamos familiarizados con los comandos. Aún así, existe la posibilidad de mandar tareas a la cola por línea de comandos sin generar scripts. Los comandos de ejecución y las opciones de estos se resumirán posteriormente.

la generación de scripts sigue una estructura de cabecera y comandos a ejecutar:

Fichero 2.5: Ejemplo.1 de script para ejecución con SGE

```
[sysadm1@frontend-0 sysadm1]\$ cat script.sh
\#!/bin/bash
\#
\#\$ -cwd
\#\$ -j y
\#\$ -S /bin/bash
\#\$ -l h_rt=0:1:0
\#
date
sleep 10
date
```

La línea -cwd sirve para indicar que la referencia al path se realiza en el directorio actual (SGE se encarga de gestionar los paths). La línea -j y, sirve para que los errores vayan a la misma salida que el resultado de ejecución del programa (generalmente son diferentes). Esto puede resultar muy útil para la ejecución remota, debido a que SGE guarda esta salida en un fichero que podremos leer

una vez terminada la ejecución de la tarea. La línea -S /bin/bash indica que el intérprete de comandos es bash. Con -l se indica el tiempo máximo de ejecución, aumenta ligeramente la eficacia. Con -l mf=<memory> se indica el volumen de memoria disponible.

Una vez se ha generado el anterior fichero de ejecución, debe mandarse la tarea a la cola para que SGE asigne recursos entre las máquinas que componen el cluster y la tarea pueda ser ejecutada.

qsub script.sh - Añade la tarea en la cola y muestra un mensaje de submit:

```
Your job 17 ("script.sh") has been submitted
```

Una vez se ha añadido la tarea en la cola, podemos monitorizar el estado de la misma, utilizando el comando **qstat**. El esquema de planificación configurado por el administrador determinará la prioridad de la nueva tarea con respecto a tareas pendientes o tareas que están siendo ejecutadas. Sin ninguna influencia del administrador, el esquema de planificación es FIFO, por lo que, la tarea será atendida según su orden de llegada. Para más información sobre el planificador de SGE se recomienda la lectura de la documentación proporcionada por Oracle [6].

qstat - Muestra el estado del proceso en la cola, generalmente le sigue *-f* para ver todas las tareas de la cola o *-j jobnumber* para ver el estado detallado de una tarea. Es una buena forma, por tanto, de conocer cuándo finaliza una tarea.

Téngase en cuenta que, para observar únicamente el estado de la tarea del ejemplo anterior, bastaría con utilizar `qstat -j 17`. La información proporcionada sigue el siguiente formato

Fichero 2.6: Ejemplo.2 Salida del comando qstat -j jobnumber

```
qstat -j 17
==============================================================
job_number:                 17
exec_file:                  job_scripts/17
submission_time:            Sun May 18 17:10:40 2012
owner:                      cmlentis
uid:                        500
group:                      cmlentis
gid:                        500
sge_o_home:                 /home/cmlentis
sge_o_log_name:             cmlentis
sge_o_path:                 /opt/gridengine/bin/lx26-amd64:/...
sge_o_shell:                /bin/bash
sge_o_workdir:              /home/cmlentis/sge
sge_o_host:                 ash25
account:                    sge
cwd:                        /home/cmlentis/sge
path_aliases:               /tmp_mnt/ * * /
merge:                      y
```

```
    mail_list:                  cmlentis@tsc.uc3m.es
    notify:                     FALSE
    job_name:                   sleep.sh
24  jobshare:                   0
    shell_list:                 /bin/bash
    env_list:
    script_file:                sleep.sh
    scheduling info:            There are no messages available
```

Además de la monitorización mediante *qstat*, podemos realizar otras acciones como la detención o eliminación de una tarea:

qhold -j jobnumber - Detiene la tarea *jobnumber*.

qdel jobnumber - Elimina de la cola la tarea *jobnumber* cuya salida por pantalla muestra el siguiente mensaje:

qdel 17

cmlentis has registered the job 17 for deletion

El comando *qdel* puede ir acompañado, además, de diferentes opciones como por ejemplo *-f* para forzar la eliminación de la tarea ante algún problema o error, o *-u user*, para realizar la eliminación de todas las tareas de un determinado usuario (*qdel -u cmlentis*).

Por último, se va a realizar un listado con las opciones más habituales y generales de SGE y que usaremos en nuestros scripts:

- -cwd: lanza la tarea en el directorio actual. Por defecto, se sitúa en el $HOME de cada usuario.
- -pe: Pararel enviroment, indica qué recursos se necesita inicializar.
- -S [shell]: campo obligatorio, indica el intérprete de comandos.
- -M mail@addr.es: se envía un correo a la dirección *mail@addr.es* al finalizar.
- -m [a\b\e]: se envía un correo a la dirección *mail@addr.es* cuando la tarea pasa por los estados: [*aborting**starting**ending*].
- -N req_name: indica el nombre de la petición, por defecto, es el nombre del script.
- -o filename: indica el fichero de salida, por defecto, es el nombre del script y termina en .o[jobnumber].

- -e filename: indica el fichero de errores del programa, por defecto, es el del script terminado en .e[jobnumber]).
- -i [host:/directorio/] filename: indica el fichero de entrada del programa que sustituye a la entrada *standard*.
- -j y: sirve para que los errores tengan la misma salida que el programa (-e es ignorado).
- -v var1[=val1][,var2[=val2],...]: carga variables de entorno. Es recomendable cargarlas en el script y no por parámetro.
- -V: carga todas las variables de entorno.
- -h: encola un trabajo en estado de parada.
- -l h_rt=x:y:z: limita el tiempo de ejecución a x horas, y minutos y, z segundos.
- -l mf=memory: especifica los requisitos de memoria sin los cuales no se ejecutará la tarea.
- -l h_vmem=memory: establece la memoria virtual por core que podrá solicitar el trabajo mandado. Es un límite estricto. Si el proceso intenta reservar más memoria, la correspondiente llamada *allocate* devolverá un error.
- -l h_fsize=filesize: especifica el tamaño máximo del fichero que el trabajo podrá escribir en disco.
- -q queue_name: especifica la cola a usar para esa tarea.
- -q queue_name@ nodename: especifica que la tarea será lanzada en nodename de la cola queue_name.

Se ha presentado una metodología que nos permite obtener un análisis paramétrico distribuido con HFSS. Esta metodología es configurable en la elección de los parámetros, en el número de estos, en el número de modificaciones de cada parámetro y en la elección del tipo de resultados obtenidos.

Una vez desarrollada la anterior metodología, será posible plantear un escenario de optimización en el que obtendremos los valores deseados de los parámetros fijados tras realizar un algoritmo de búsqueda.

Capítulo 3

Simulación paramétrica distribuida con HFSS

Por las motivaciones descritas en el capítulo 1 y, ante la necesidad de un uso eficiente de los recursos que se disponen, se ha desarrollado una metodología para realizar análisis paramétricos distribuidos en el cluster de computación científica del Grupo de Radiofrecuencia.

En la Figura 3.1 se muestra el sistema completo, integra la herramienta de optimización y la simulación parámetrica distribuida en HFSS. Es importante resaltar que la realimentación en el estado que representa la simulación paramétrica, se debe a la espera que se realiza hasta que todas las tareas han finalizado y a la espera que se requiere por la re-simulación de los proyectos con posibles errores.

La metodología desarrollada será usada en el proceso de optimización de estructuras electromagnéticas. Esta herramienta de optimización se basa en un algoritmo genético y ha sido desarrollada dotando de flexibilidad al sistema para mejorar la experiencia del usuario. Se ha seleccionado un algoritmo genético, puesto que, es intrínsecamente paralelo, esto permite explorar el espacio de búsqueda y analizar en paralelo cada uno de los puntos seleccionados de dicho espacio (ver capítulo 4).

A continuación, se va a desarrollar en profundidad la metodología propuesta, sobre la que podemos diferenciar 3 fases: Preproceso, resolución y post-proceso. Ver Figura 3.2.

Para ilustrar la metodología propuesta se va a realizar a modo de ejemplo un análisis paramétrico sobre un filtro de resonadores dieléctricos en guiaonda [7] [8]. Este filtro está compuesto por 4 cavidades con 5 irises inductivos.

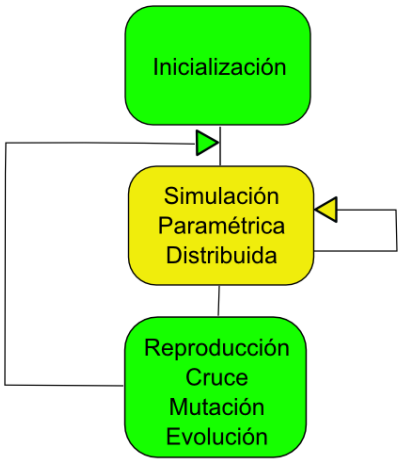


Figura 3.1: Diagrama de flujo sistema completo

El uso creciente de estos dispositivos se basa en la reducción de masa y volumen del filtro respecto al filtro de cavidad metálica, además de mejorar la estabilidad térmica en aplicaciones de alta potencia. Este filtro tiene varios parámetros que definen su geometría, que como podemos ver, es simétrica; por ello, variaremos al mismo tiempo los irises de acoplo de dos cavidades y la longitud en el eje horizontal del slot central.

En este capítulo se muestra una metodología que ha sido validada por el artículo [9], presentado en el Simposium Nacional de la Unión Científica Internacional de Radio, URSI 2012 [10].

3.1. Preproceso

La fase de preproceso consiste en la generación de un determinado número de proyectos para su posterior análisis, donde cada proyecto, está configurado con unos parámetros prefijados por el diseñador y que adquieren los valores que el usuario decide, tal y como se puede apreciar en Figura 3.2 (la primera caja en azul).

De esta forma, se consigue una población de análisis que en conjunto constituyen el análisis paramétrico que se desea realizar. Esta metodología proporciona mucha flexibilidad, ya que el diseñador puede determinar el número de parámetros de la estructura a modificar y seleccionar cada uno de ellos. Además, el diseñador puede decidir el número de modificaciones sobre cada uno de estos parámetros convirtiendo su análisis paramétrico en N análisis. Este N vendrá dado por el

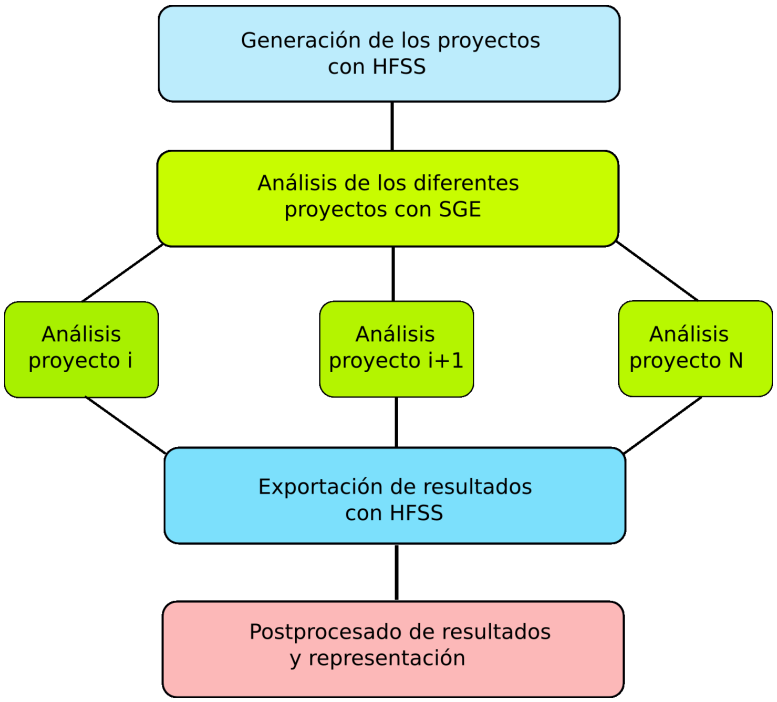


Figura 3.2: Diagrama de flujo en el análisis de sensibilidad.

algoritmo de generación: $N = \prod_{k=1}^{M} \#valores_k$, donde M sería el número de parámetros y $\#valores_k$ el número de valores que puede tomar cada parámetro. En este caso se emplea el algoritmo de fuerza bruta.

Todos los proyectos son generados a partir de un proyecto raíz en el que se parte de un estado inicial y sobre el cual se modifican sus parámetros para obtener el correspondiente subproyecto. De esta forma, se consigue que cada proyecto generado sea independiente del resto, con lo que no se presentan limitaciones a la hora de realizar el análisis del proyecto.

La generación del subproyecto i-ésimo ha sido realizada mediante script de VBasic

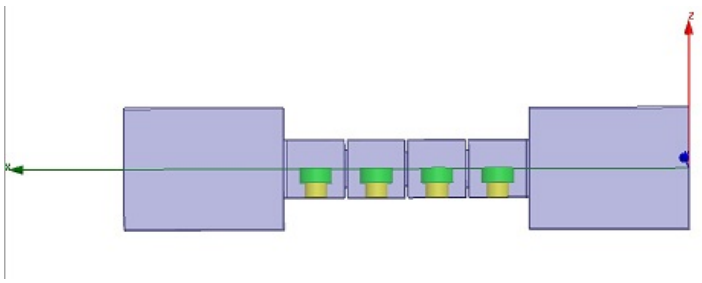


Figura 3.3: Filtro diléctrico de resonadores en guiaonda.

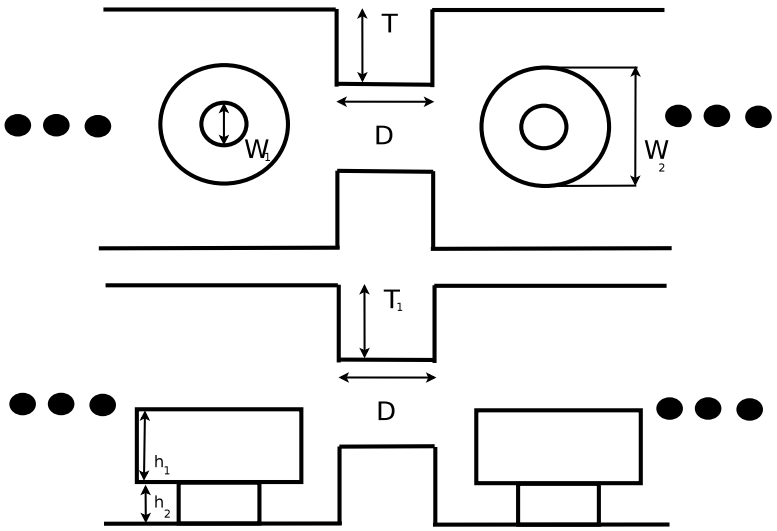


Figura 3.4: Parámetros del filtro dieléctrico.

en HFSS. Además, y como se ha comentado en la introducción, el entorno de trabajo es Linux, siendo la solución extrapolable a Windows con mínimos cambios en el script que controla el gestor de recursos.

Este lenguaje de scripting presenta una importante problemática: requiere que la interfaz gráfica esté activa en la ejecución de scripts. Es por ello que, se ha dividido la metodología en tres fases: preproceso, resolución y postproceso; donde la fase de resolución se encarga del análisis de proyectos caracterizados electromagnéticamente y con un análisis setup definido, sin necesidad de utilizar la interfaz gráfica.

El optimizador debe dirigir la simulación paramétrica a través de una interfaz, indicando los valores que adquieren los parámetros en cada combinación parámetrica bajo análisis. El usuario debe poder indicar, entre otras opciones, qué parámetros desea optimizar, cómo se codifican o el espacio de búsqueda en el que se localizará la solución óptima (todo ello se describe con mayor nivel de detalle en el capítulo 5).

Para generar los subproyectos se escriben N ficheros *.vbs* a través de la llamada iterativa a la función *imprimirVbs.m*, tal y como muestra el diagrama de flujo de la Figura 3.5). Cada fichero *.vbs* posee el valor que adquiere cada parámetro, además de información relativa al tipo de análisis que se desea realizar. Esta información comprende, entre otras opciones.

- Nombre del proyecto a optimizar.

- Frecuencia a la cual se ha diseñado el dispositivo.
- Configuración del tipo de análisis a realizar por HFSS. En el caso de no realizarse ningún barrido paramétrico, este campo se indicará con el caracter vacío ” y en el caso de realizarse un barrido paramétrico se indicará con el string *Sweep*.
- Frecuencia de inicio en un análisis paramétrico.
- Frecuencia de final en un análisis paramétrico.
- Frecuencia de paso en un análisis paramétrico.

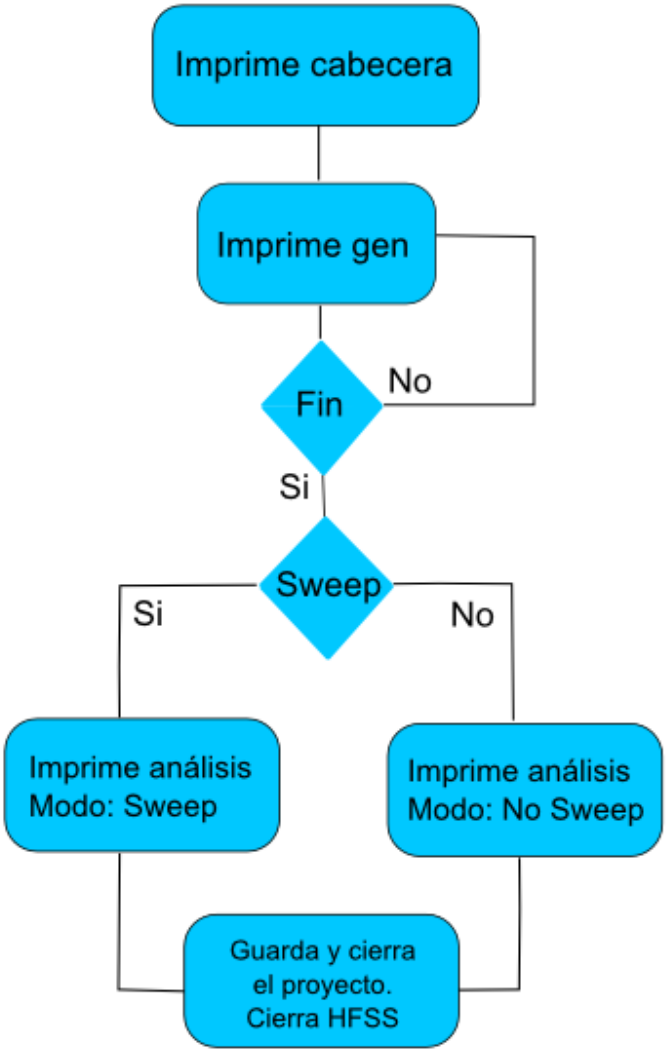


Figura 3.5: Diagrama de flujo de la función imprimirVbs.m

Nótese que, como se comentó anteriormente, existe total libertad de decisión a la hora de seleccionar los parámetros de la estructura, así como el número de ellos, sin existir ninguna limitación.

La ejecución de los N ficheros *.vbs* genera N subproyectos, donde cada subproyecto asigna valores a los parámetros fijados. La función *imprimirVbs.m* se encarga de generar, por tanto, un fichero *.vbs* distinto para cada individuo, asignándose en cada fichero los valores de los parámetros pertenecientes al individuo, tal y como se muestra en la Figura 3.6.

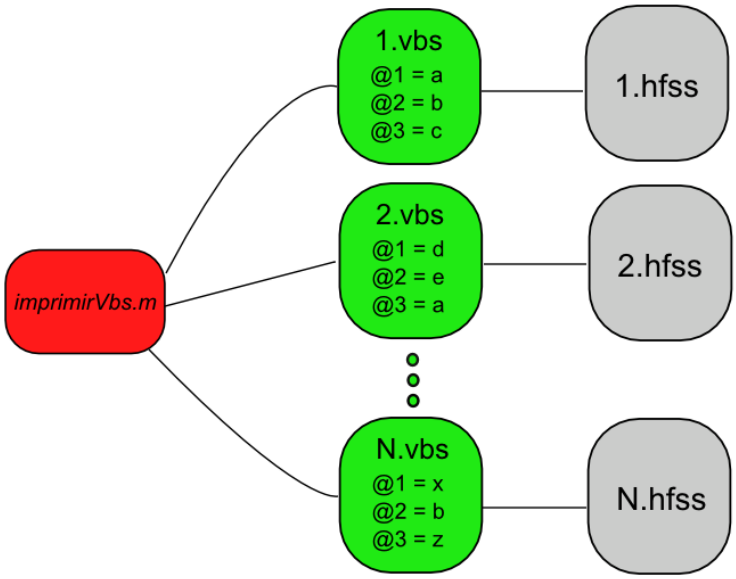


Figura 3.6: Esquema de ficheros del estado Preproceso

A continuación, se detalla la función *imprimirVbs.m*, cuyo diagrama de flujo se representa en la Figura 3.5:

En primer lugar, se imprime la cabecera del fichero *.vbs* (ver código 3.1). Esta cabecera es común a todos los subproyectos generados y en ella se declaran las variables y objetos usados por el script.

Fichero 3.1: imprimirVbs.m: Imprime cabecera

```
1 function fichero = imprimirVbs(genes,individuo,nombreProyecto,frecuencia,modo
      ,fini,ffin,fpaso)

  fichero = [num2str(individuo.ident) '.vbs']
  fout = fopen(fichero,'wt');

6 fprintf(fout,'Dim oAnsoftApp \n');
  fprintf(fout,'Dim oDesktop \n');
  fprintf(fout,'Dim oProject \n');
  fprintf(fout,'Dim oDesign \n');
```

```
   fprintf(fout,'Dim oEditor \n');
11 fprintf(fout,'Dim oModule \n');
   fprintf(fout,'\n');
   fprintf(fout,'Dim directorioS \n');
   fprintf(fout,'Dim args \n');
   fprintf(fout,'Dim local_var_array \n');
16 fprintf(fout,'Dim nom \n');

   fprintf(fout,'\n');
   fprintf(fout,'Dim units, delta, npasos, lambda, fini, ffin, fpaso \n');
   fprintf(fout,'\n');
21
   fprintf(fout,'units = "GHz" \n');
   fprintf(fout,'delta = 0.02 \n');
   fprintf(fout,'npasos = 4 \n');
   fprintf(fout,'lambda = 0.4 \n');
26 fprintf(fout,'dir = "./" \n');
   fprintf(fout,'directorioS = dir & "paramS" \n');
   fprintf(fout,'\n');
   fprintf(fout,'Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInterface")
       \n');
   fprintf(fout,'Set oDesktop = oAnsoftApp.GetAppDesktop()\n');
31 fprintf(fout,'\n');

   a1 = 'oDesktop.OpenProject "'
   a2 = nombreProyecto;
   a3 = '.hfss" \n';
36 cadenaA = [a1 a2 a3];
   fprintf(fout,cadenaA);
   fprintf(fout,'\n');

   b1 = 'Set oProject = oDesktop.SetActiveProject("';
41 b2 = nombreProyecto;
   b3 = '") \n';
   cadenaB = [b1 b2 b3];
   fprintf(fout,cadenaB);

46 fprintf(fout,'Set oDesign = oProject.SetActiveDesign("HFSSDesign1") \n');
   fprintf(fout,'Set oModule = oDesign.GetModule("AnalysisSetup") \n');
   fprintf(fout,'\n');
```

Posteriormente, se abre un fichero *.vbs* en modo escritura cuyo nombre está almacenado en el campo *ident* del individuo (capítulo 4) pasado por parámetro. Una vez obtenido el manejador del fichero se procede a escribir con la información de ese individuo y con la información genética, el fichero VBScript.

La escritura se realiza línea a línea con la función *fprintf*, que recibe el manejador del fichero y el *string* a escribir. Se finaliza la línea con un retorno de carro `\n`.

En esta primera parte, se declaran las variables y se crean los objetos necesarios para el VBScript tal y como se indicó en el capítulo introductorio.

Después se abre el proyecto cuyo nombre se pasa por parámetro y se crea el objeto *oProyect*. La escritura se realiza gracias a la concatenación de *strings*, sustituyéndose el campo genérico: 'nombre de proyecto', por el proyecto que el

usuario decide optimizar. A continuación, se crean objetos de los siguientes niveles jerárquicos.

El fichero tiene puntos críticos que son propios a cada individuo y que no comparte el conjunto (valor de los parámetros). Esta información debe escribirse con los argumentos que recibe la función, tal y como se detalla en el código 3.2.

Fichero 3.2: imprimirVbs.m: Imprime Gen

```
for s=1:length(genes)

  gen1 = genes(s).valor;
  numbits =gen1.nbits;
  valorParam = cromosoma.valor;
v = valorParam(1:numbits)
  valor = (gen1.centro) + (gen1.spam)*bin2dec(v)
  nombre = gen1.variable;


c0='oDesign.ChangeProperty Array("NAME:AllTabs", Array("NAME:LocalVariableTab
    ", Array("NAME:PropServers", _ \n'
  c1=' "LocalVariables"), Array("NAME:ChangedProps", Array("NAME:'
  c2 = nombre;
  c3= '", "Value:=", "';
  c4= [num2str(valor) gen1.unidad];
c5='"))))) \n ';
  cadena1 = [c0 c1 c2 c3 c4 c5];
  fprintf(fout,cadena1);
  fprintf(fout,'\n');
  fprintf(fout,'\n');

  end
```

Otro de los puntos críticos en la generación del fichero *.vbs* es el valor de los paramétros, los cuales fija el optimizador (capítulo 4). Se añadirá una línea al fichero por cada parámetro de interés. La información que proporciona el optimizador es: nombre del parámetro a optimizar, unidades de medida o rango de valores sobre el que se realiza la búsqueda.

Debe tenerse en cuenta cómo están definidos en el proyecto los parámetros, pueden ser variables de proyecto o variables locales. En este caso, se han definido los parámetros como variables locales por lo que se usa *LocalVariableTab*, en caso contrario, se usaría *ProjectVariableTab*.

En las primeras líneas del código 3.2, se obtiene el nombre del parámetro *i-esimo* y se decodifica la información para obtener su valor. Después, se genera a través de la concatenación de strings, una llamada a la función *oDesign.ChangeProperty*, que recibe el nombre del parámetro y su valor junto con las unidades de medida. Se genera una línea de este tipo por cada parámetro a optimizar.

Lo siguiente a definir es la configuración del análisis (código 3.3), donde se fija únicamente la frecuencia, puesto que el refinamiento del mallado se ha dejado fijo, cualquier cambio en dicho refinamiento debería realizarse aquí.

Fichero 3.3: imprimirVbs.m: imprime análisis

```
b1 = '  oModule.InsertSetup "HfssDriven", Array("NAME:Setup1", "Frequency:=",
    "';
b2 = frecuencia;
b3 = '", "PortsOnly:=", _ \n';
cadena2 = [b1 b2 b3];
fprintf(fout,cadena2);
fprintf(fout,'  false, "MaxDeltaS:=", delta, "UseMatrixConv:=", false, "
    MaximumPasses:=", npasos, "MinimumPasses:=", _ \n');
fprintf(fout,'  1, "MinimumConvergedPasses:=", 1, "PercentRefinement:=", 30,
    "IsEnabled:=", _ \n');
fprintf(fout,'  true, "BasisOrder:=", 1, "UseIterativeSolver:=", false, "
    DoLambdaRefine:=", _ \n');
fprintf(fout,'  true, "DoMaterialLambda:=", true, "SetLambdaTarget:=", true,
    "Target:=", _ \n');
fprintf(fout,'  lambda, "UseMaxTetIncrease:=", false, "PortAccuracy:=", 2, "
    UseABCOnPort:=", _ \n');
fprintf(fout,'  false, "SetPortMinMaxTri:=", false, "EnableSolverDomains:=",
    false, "ThermalFeedback:=", _ \n');
fprintf(fout,'  false, "NoAdditionalRefinementOnImport:=", false) \n');
fprintf(fout,'\n');
```

El código 3.3 es común a todos los subproyectos generados. En él, se fija la frecuencia de funcionamiento a la que se diseña el dispositivo. Además, se fijan las condiciones de refinamiento.

En el caso de no realizarse barrido frecuencial, el código anterior basta para obtener el resultado de ese punto de frecuencia para el parámetro de análisis.

Fichero 3.4: imprimirVbs.m: Imprime Análisis

```
if (isequal(modo,'Sweep') == 1)
fprintf(fout,'oModule.InsertFrequencySweep "Setup1", Array("NAME:Sweep", "
    IsEnabled:=", true, "SetupType:=", _ \n');

e1= '"LinearStep", "StartValue:=", "';
e2= fini;
e3= '", "StopValue:=", "';
e4=ffin;
e5='", "StepSize:=", _ \n';
cadena7 = [e1 e2 e3 e4 e5];
fprintf(fout,cadena7);

f0='  "';
f1= fpaso
f2= '", "Type:=", "Interpolating", "SaveFields:=", false, "InterpTolerance:="
    , _ \n';
cadena8 = [f0 f1 f2];
fprintf(fout,cadena8);
fprintf(fout,'  0.5, "InterpMaxSolns:=", 250, "InterpMinSolns:=", 0, "
    InterpMinSubranges:=", 1, "ExtrapToDC:=", _ \n');
fprintf(fout,'  false, "InterpUseS:=", true, "InterpUsePortImped:=", false, "
    InterpUsePropConst:=", _ \n');
fprintf(fout,'  true, "UseDerivativeConvergence:=", false, "
    InterpDerivTolerance:=", 0.2, "UseFullBasis:=", _ \n');
```

```
    fprintf(fout,'  true) \n');
    end
```

Si el usuario ha establecido un análisis con barrido frecuencial, este habrá introducido la cadena *Sweep* como argumento (ver código 3.4). Se realiza una comparación del argumento con la función *isequal* y si se ha seleccionado el modo *Sweep* se escribe en el fichero una línea para realizar una llamada a la función *oModule.InsertFrequencySweep*. En esta línea se debe especificar la frecuencia de inicio, la frecuencia fin y la frecuencia de paso. Estos valores se definidos en la llamada a la función *imprimirVbs.m*.

Para finalizar, se debe guardar el proyecto con el nuevo nombre y cerrarlo, operación que se realiza a semejanza a como se realizó la apertura del mismo (ver código 3.5).

Fichero 3.5: imprimirVbs.m: Guarda y cierra HFSS

```
    a1= 'oProject.SaveAs dir & "';
    a2= num2str(cromosoma.ident);
    a3= '.hfss", true \n';
 4  cadena3 = [a1 a2 a3];
    fprintf(fout,cadena3);

    d1= 'oDesktop.CloseProject "';
    d2= num2str(cromosoma.ident);
 9  d3= '" \n';
    cadena4 = [d1 d2 d3];
    fprintf(fout,cadena4);

    fprintf(fout,'oProject.Close \n');
14  fprintf(fout,'\n');
```

Es importante recordar, que se podría caracterizar un proyecto de forma completa desde su inicio, definiendo la estructura deseada, el material, el análisis a realizar o el report de salida. En definitiva, podemos desarrollar un VBScript con cualquiera de las opciones que nos proporciona la interfaz gráfica de HFSS.

Antes de detectar la necesidad de realizar numerosos cambios en el VBScript se diseñó el preproceso con un solo fichero *.vbs*. Este fichero recibía como argumento todos los parámetros a modificar y se iban haciendo sucesivas llamadas del mismo con distintos valores de los parámetros en función del individuo objeto de análisis. Esta nueva metodología aumenta la cantidad de ficheros generados pero no afecta significativamente al uso de espacio en disco duro puesto que, son ficheros de tamaño menor.

Podemos usar los parámetros almacenados en la variable definida como *args* para modificar la estructura de análisis a nuestro placer. Este punto es crítico en el

sistema ya que es justo en este punto del código donde fijamos los parámetros de la estructura al valor pasado por parámetro.

El uso de argumentos en un VBScript se realiza como se detalla:

Set args = AnsoftScript.arguments

La anterior línea de código obtiene el conjunto de argumentos pasados por parámetros con una llamada del tipo: *nombre.vbs @1 @2 @3 ... @N*, todos los parámetros se almacenan en el array args.

Una vez obtenidos los N ficheros *.vbs* , se generan los N subproyectos con la ejecución de los VBScripts por línea de comandos con la función *runscript* tal y como se muestra a continuación:

Fichero 3.6: generaHFSS.sh

```
1 #!/bin/ksh

  i=1
  while [ $i -le $1 ]
  do
6 hfss -runscript $i.vbs
  i=$(($i+1))
  done
```

De esta manera, solo necesitamos realizar una llamada a *generaHFSS.sh N*, donde N es el número de ficheros *.vbs* generados y que están etiquetados con enteros para realizar una identificación unívoca. Tras la generación de los N subproyectos, se pasa a la fase de resolución. Fase en la cual se realiza el análisis en paralelo de los subproyectos generados. Recuérdese que HFSS usa el Método de Elementos Fínitos (ampliamente estudiado en la biografía [1] y [11]), para obtener las soluciones de campo.

El análisis dependerá en términos de tiempo de ejecución y carga computacional del grado de refinamiento establecido para obtener la solución. Los parámetros que establecen el refinamiento son los siguientes:

- Lambda Refinement: es el proceso de refinamiento de la malla inicial, basado en la longitud de onda y en el propio material de la estructura. Se define un objetivo que depende de una función de un determinado orden. Para el caso de un modelo Driven, que es el caso que se trata, se usa por defecto una función de primer orden y con un objetivo del 0.333, esto significa que HFSS refinará la malla hasta que la mayoría de los elementos tengan aproximadamente 1/3 de longitud de onda.

- Maximum Number of Passes: se trata del número máximo de ciclos de refinamiento que se van a realizar, si se alcanza este número el análisis se detiene, si no se alcanza es porque el criterio de convergencia hará parar el análisis.Esto supone un criterio de parada.
- Maximum Delta S Per Pass: este valor simboliza la variación que experimentan los parametros [S] de un paso al siguiente, con lo que si esta variación es menor al número que hemos fijado, el análisis se detiene.
- Maximum Refinement Per Pass: este valor determina cuántos tetaedros serán añadidos en cada iteración del proceso de refinamiento. El tetaedro con mayor error será redefinido.
- Otra opciones son fijar el máximo número de tetaedros que se pueden añadir por paso o el mínimo número de ciclos que se deben realizar antes de que el análisis finalice. Además, HFSS da la oportunidad de fijar criterios de parada propios a parte de los anteriormente expuestos.

Visto lo anterior, se puede comprender mejor como actúa la función InsertSetup, que sigue la estructura *InsertSetup <SetupType>, <AttributesArray>*. Esta función tiene como argumentos los parámetros que se acaban de definir, los más relevantes son los siguientes:

- NAME: nombre asociado al *análisis setup*.
- Frequency: frecuencia a la que se ha diseñado el dispositivo.
- DoLambdaRefine: en nuestro script está marcado con el valor *true* y sirve para indicar a HFSS que debe realizar el citado proceso de refinamiento sobre la malla inicial. Se puede establecer la longitud de onda a la que HFSS refinará el mallado.

Como se indicó anteriormente, cualquier modificación de las condiciones de mallado se realizan en el fichero *imprimirVbs.m*, función donde se han fijado. Se podría optar por pasar por parámetro dichas condiciones, simplemente se debe escribir el fichero con la metodología vista.

Uno de los inconvenientes de generar N subproyectos a partir de un proyecto raíz, es el espacio en disco, ya que si se desea un elevado número de análisis se generan un elevado número de proyectos que debemos almacenar en disco. De esta forma, se incrementa la capacidad en disco necesaria para llevar a cabo el análisis. En particular, los resultados de los proyectos generados con esta metodología con

el ejemplo propuesto, ocupan del orden de 12 Mb por proyecto. En este caso, no resulta relevante, pero sí que es un factor a tener en cuenta si se deseara un elevado número de simulaciones o si los proyectos fuesen más complejos. Es decir, la solución ocupa el mismo tamaño que si se realizara el análisis en secuencial, pero al dividir el proyecto en varios, duplicamos esa información. Además, el elevado número de ficheros en diversos formatos que se están almacenando, se deben gestionar y manipular en las posteriores etapas de diseño, lo cual aumenta la complejidad.

3.2. Resolución

Para realizar la simulación de todos los proyectos generados en la fase preproceso de forma distribuida, se utiliza el sistema de gestión de colas SGE descrito en el capítulo 2. Entre las opciones presentadas en dicho capítulo, existe una que resulta especialmente interesante para nuestro propósito, pues permite enviar todas las tareas de forma simultánea con una sola llamada. Esta opción, se denomina vector de tareas (*job array*) y ha sido ilustrada mediante ejemplos sencillos.

Mediante esta opción, la variable de entorno (`SGE_TASK_ID`) toma valores dentro del rango especificado en `-t límiteInferior-límiteSuperior`, simplificando nuestra interacción con la cola, ya que los proyectos a analizar se identifican de forma unívoca por los valores que adquiere.

Uno de los problemas que se han encontrado en esta fase es el desconocimiento por parte del usuario sobre el estado de la simulación paramétrica, es decir, lanzamos en paralelo el análisis de los N subproyectos pero desconocemos el momento exacto en el que estos finalizan. Para ello, se ha introducido en el script maestro que realiza la resolución, un mecanismo de espera activa que será sustituido en líneas futuras por un mecanismo de espera por eventos:

Fichero 3.7: Mecanismo de espera en el gestor de colas

```
qsub cola.sh

flag=1
while [ $flag -eq "1" ]
do
qstat >> qstat.txt

        if [ -s qstat.txt ]
        then
        flag=1
        sleep 90
        else
        flag=0
        fi
rm -r qstat.txt
```

```
done
```

En primer lugar, se mandan las tareas a la cola con el comando *qsub*. Después, se inicializa un flag con valor '1' e iteramos un bucle mientras el flag conserva este valor. Dentro del bucle escribimos en el fichero *qstat.txt* la salida del comando *qstat*, mientras que el fichero no esté vacío el comando *qstat* sigue monitorizando tareas pendientes y el flag no cambia su valor. En el momento en el que el fichero está vacío se detecta que el análisis ha finalizado y se modifica el valor del flag.

El comando *sleep* sirve para monitorizar cada $90seg$ el estado, puesto que el planificador no nos dará una nueva respuesta del comando *qstat* hasta no iniciar un nuevo intervalo de planificación.

Por último, es importante borrar el fichero en cada iteración puesto que las escrituras se apilan.

El script *cola.sh* que veremos en el siguiente código manda las tareas a la cola:

Fichero 3.8: cola.sh

```
#!/bin/bash
#$ -V
#$ -cwd
#$ -j y
#$ -S /bin/bash

#$ -N hfss_SD
#$ -q all.q
#$ -pe orte 8
#$ -l mf=30G
#$ -l h_vmem=4G
#$ -t 1-16

NofP=1
REGISTRO=registry$JOB_ID.txt
cat > $REGISTRO << EOF

\$begin 'Config'
'HFSS/Preferences/NumberOfProcessorsDistributed'=$NofP
'HFSS/Preferences/NumberOfProcessors'=8
'HFSS/Preferences/MemLimitHard'=31000000
'HFSS/Preferences/MemLimitSoft'=19000000
'HFSS/Preferences/UseHPCForMP'=1
'HFSS/Preferences/HPCLicenseType'=pool
\$end 'Config'
EOF

FILEHFSS=$SGE_TASK_ID.hfss

hfss -ng -WaitForLicense -BatchSolve -machinelist num=1 -batchoptions
    $REGISTRO $FILEHFSS

file=$SGE_TASK_ID
re=".hfssresults"
total="$file$re"
band=1
```

```
   while [ band -eq 1 ]
   do
   find ./resultados/$total "M2" >> encuentra.txt
39 fichero=`sed '1d' encuentra.txt`
   rm -r encuentra.txt
   echo $fichero >> encuentra.txt
   grep -c "\$end 'soln'" $fichero >> escribo.txt

44                 for line in $(cat escribo.txt)
                   do
                           if [ "$line" = "1" ] || [ "$line" = "2" ] || [ "$line
                               " = "3" ] || [ "$line" = "4" ]
                           then
                           echo victoria
49                         band=0
                           else
                           echo fracaso
                   rm -r $file.log
                   rm -r $file.hfss.lock
54                 rm -r $JOB_NAME.o$JOB_ID
                   rm -r $JOB_NAME.e$JOB_ID
                   rm -r registry$JOB_ID.txt

                   NofP=1
59                 REGISTRO=registry$JOB_ID.txt
                   cat > $REGISTRO << EOF

                   FILEHFSS=$SGE_TASK_ID.hfss
                   rm -r $SGE_TASK_ID.hfss.lock
64                 hfss -ng -WaitForLicense -BatchSolve -machinelist num=1 -
                       batchoptions $REGISTRO $FILEHFSS

                   fi
                   done

69 rm -r encuentra.txt
   rm -r escribo.txt
   done
```

La línea -V carga todas las variables actuales, -cwd sirve para indicar que la referencia al path se realiza en el directorio actual (SGE se encarga de gestionar los paths). La línea -j y, sirve para que los errores vayan a la misma salida que el resultado de ejecución del programa (generalmente son diferentes). Esto puede resultar muy útil para la ejecución remota, debido a que SGE guarda esta salida en un fichero que podremos leer una vez terminada la ejecución de la tarea. La línea -S /bin/bash indica que el intérprete de comandos es bash.

Para indicar el nombre de la petición se usa *-N < nombre_peticion >*. Los ficheros de salida y error comienzan con el mismo nombre. Posteriormente, se especifica con *-q all.q* que el maestro hará uso de todos los nodos esclavos y, con *-pe orte 8* los recursos que es necesario inicializar en el entorno paralelo. Para finalizar, indicamos el volumen de memoria física y virtual disponible con *-l mf* y *-l h_vem*.

La línea 12 es de especial relevancia:

#$-t 1-N

Con esta línea indicamos al gestor de colas que se van a distribuir en paralelo N trabajos, la variable *SGE_TASK_ID* adquirirá progresivamente todos los valores desde 1 hasta N. Gracias a la identificación unívoca diseñada, se ejecutarán todos los subproyectos generados puesto que fueron nombrados con los mismos valores que adquiere la variable *SGE_TASK_ID*.

Las líneas 14-25 escriben algunos parámetros de configuración de HFSS en el registro. En este caso, se han solicitado 8 procesadores de un solo nodo del cluster para realizar el trabajo etiquetado con la variable *SGE_TASK_ID*. Además, se indican límites de memoria requerida para realizar dicho trabajo.

Las líneas 28-32 ejecutan por línea de comandos el subproyecto `&(SGE_TASK_ID).hfss`. Esta ejecución se realiza por medio de *-BatchSolve* por lo que el análisis se realiza sin necesidad de activar la interfaz gráfica. Además, va acompañado de la opción *-WaitForLicense*, que como se vió en el capítulo introductorio, sirve para esperar licencias libres de HFSS para lanzar la ejecución.

Pueden existir problemas en el análisis de una esctructura congelándose este, en la cola. Para ello, se ha implementado un mecanismo que reinicie la ejecución de HFSS en el caso de que se detecte un comportamiento anómalo. Tras el estudio de los ficheros que genera HFSS en la simulación, se ha observado que existe un fichero maestro que guía las diferentes fases por las que pasa este proceso.

HFSS genera en su análisis, ficheros que indican la fase en la que se encuentra la simulación, entre otros, ficheros con las soluciones de campo. Resulta intuitivo, por tanto, diseñar un mecanismo que compruebe en primer lugar, la existencia del fichero de interés, y posteriormente, la escritura en el mismo, del comando que indica el final no solo de dicha fase, sino en nuestro caso, el final de la simulación, con o sin errores.

Este mecanismo realiza una espera que finaliza cuando el flag *band* cambia de valor. Con el comando *find* se busca un fichero que contenga los caracteres $'M2'$ y se copia su nombre en el fichero *encuentra.txt*. Esto se realiza porque el nombre del fichero que contiene los caracteres fijos $'M2'$, contiene también, caracteres que entre ejecuciones consecutivas difieren entre sí.

A continuación, se elimina la primera línea del fichero *encuentra.txt* puesto que también almacena el directorio *./*, una vez que tenemos en el fichero únicamente el nombre se almacena en la variable *fichero*.

La existencia de este fichero garantiza que HFSS se encuentra en ese estado concreto de la simulación, pero no garantiza su final, puesto que puede estar

ejecutándose precisamente dicho estado. Es por ello, que no basta con comprobar la existencia del fichero, sino que se debe leer el mismo, comprobando que HFSS ha escrito el comando que indica su finalización, comando `$end`.

La línea 42 ($grep - c"\$end'soln'"\$fichero >> escribo.txt$) busca la cadena de caracteres $\$end'soln'$ en el fichero $\$fichero$ y vuelca el número de coincidencias en el fichero *escribo.txt*.

Las líneas 44-49 leen el fichero *escribo.txt* para comprobar el número de coincidencias existentes, y si existe alguna, se detecta que la ejecución de la simulación discurre sin problemas por lo que se modifica el valor del flag y se finaliza la espera. En caso contrario (línea 50), borramos los ficheros generados reseteando el mecanismo y volvemos a repetir el proceso, configurando el registro y llamando de nuevo a HFSS.

A pesar de monitorizar directamente el estado de la simulación en HFSS, el mecanismo anterior solo asegura que HFSS esté ejecutándose con normalidad. Otro problema detectado es que HFSS realice su flujo de ejecución con normalidad pero con errores visibles en los ficheros *.log* de salida.

En este caso, HFSS ha realizado su flujo de ejecución correctamente, pero ha informado al usuario de la existencia de errores que le impiden realizar una simulación completa y sin fallos.

Un posible error que se podría encontrar, es un error de licenciamiento, este se debe al acceso simultáneo de dos procesos compitiendo por las licencias disponibles de HFSS. En este caso, el comando -*WaitForLicense* no resuelve el problema, puesto que este comando hace esperar al proceso hasta que una licencia está libre, y no plantea solución cuando dos recursos acceden en el mismo instante de tiempo a las licencias libres.

El anterior problema se resuelve con la monitorización de los archivos *.log*. La idea general que se propone es la búsqueda de errores en los ficheros *.log* de cada subproyecto generado (recuérdese que cada subproyecto analizado genera sus propios directorios y ficheros de salida, log, error, etcétera). Detectado el error, se lanza de nuevo el subproyecto al gestor de colas para su análisis de forma exclusiva.

Fichero 3.9: monitorización de los ficheros .log

```
while [ $cont -le $contador ]
do
        if [ ! -f $cont.log ]
4       then
        rm -r *.lock
        export VAR=$cont
        qsub cola2.sh
```

```
        sleep 300
        else
        grep -c 'license error' $cont.log >> si.txt
                for line in $(cat si.txt)
                do
                        if [ $line -eq "1" ] || [ $line -eq "2" ] || [ $line
                            -eq "3" ] || [ $line -eq "4" ]
                        then
                        export VAR=$cont
                        rm -r $cont.log
                        qsub cola2.sh
                        sleep 300
                        fi
                done
        fi
  cont=$(($cont+1))
  rm -r si.txt
done
  cont=1
```

El código anterior comprueba, en primer lugar, la existencia de los ficheros *.log*. Las comprobaciones se realizan fichero a fichero, siendo cada fichero *.log* el asociado a un subproyecto. Esta comprobación se realiza con `$if [ ! -f $cont.log ]` donde $*cont* es el identificador del subproyecto *i-ésimo*.

En el caso de no existir el fichero *.log*, se vuelve a lanzar el trabajo a la cola, en este caso de forma exclusiva. Para ello, se ha implementado el script *cola2.sh*, este script no dispone la opción -t sino que coge la variable definida por *export VAR*, que contiene el identificador del subproyecto que no ha generado ningún tipo de log y lo manda a SGE para su análisis.

En el caso de que exista el fichero *.log*, se buscan, con el comando grep anteriormente visto, coincidencias con la cadena *license error*. Si se encuentran coincidencias, se vuelve a lanzar la tarea al sistema gestor de colas, y si no hay coincidencias, se da por válida la simulación.

Los mecanismos vistos son de gran utilidad cuando el número de análisis crece exponencialmente. De esta forma, no se requiere una participación activa del usuario, cuyo único cometido, será establecer el análisis paramétrico deseado.

3.3. Postproceso

Una vez distribuidos los análisis a las distintas máquinas mediante SGE y obtenida la resolución de la estructura para los parámetros prefijados en cada análisis, comienza la fase de postproceso, que se subdivide nuevamente en dos partes. En la primera, se accede a todos los proyectos analizados para obtener su report, en

la nomenclatura de HFSS; y en la segunda fase de postproceso, se analizan estos resultados y se representan.

En la primera fase del postprocesado se obtienen los parámetros [S]. Estos se pueden obtener para la frecuencia a la cual se ha diseñado el dispositivo o para un rango de frecuencias, utilizando el modo *Sweep*. Con esta metodología se puede obtener cualquiera de los posibles reports que dispone HFSS, como por ejemplo, el diagrama de radiación de una antena (si la estructura bajo análisis fuese una antena). Una vez obtenida la solución de los parámetros [S] (en nuestro caso la magnitud en dB), se exporta en formato csv toda la información. Cada una de las simulaciones exporta su fichero de resultados, siempre manteniendo la correspondencia entre estos resultados y el valor de los parámetros prefijados en la fase preproceso.

En la segunda fase de postprocesado se utiliza Matlab [12] u Octave, por la flexibilidad que nos aportan al procesar los datos obtenidos. Además, facilitan el procesado de los ficheros csv, pudiendo transformar su contenido a estructuras de datos fácilmente accesibles.

En la primera fase de postprocesado, hacemos una llamada al script ./estimados.sh. En este script se realizan N llamadas a HFSS, una por cada subproyecto generado y analizado.

Fichero 3.10: estimados.sh

```
#!/bin/ksh

m=16
n=1

        while [ $n -le $m ]
        do
        hfss -runscript proceso.vbs -scriptargs "$n"
        n=$(($n+1))
        done
```

El VBScript *proceso.vbs* recibe como único parámetro el identificador del subproyecto, solo necesitamos esta información para abrir el subproyecto y obtener los resultados requeridos.

Al igual que los ficheros generados en la fase de preproceso, *procesado.vbs* comienza declarando las variables y asignando valores a esas variables. Una vez se han establecido los objetos de los niveles jerárquicos correspondientes, se implementa la funcionalidad de esta primera fase de postprocesado.

Fichero 3.11: proceso.vbs: 1

```
Dim oAnsoftApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor
Dim oModule
Dim directorioS
Dim args
Dim local_var_array
Dim nom
Dim units, delta, npasos, lambda

units = "GHz"
delta = 0.04
npasos = 5
lambda = 0.4
dir = "./"
directorioS = dir & "paramS"
```

En esta primera parte de *proceso.vbs*, se declaran las variables de los objetos que serán creados y variables que se usarán a lo largo del VBScript, como por ejemplo, el directorio de trabajo.

Fichero 3.12: proceso.vbs: 2

```
Set oAnsoftApp = CreateObject("AnsoftHfss.HfssScriptInterface")
Set oDesktop = oAnsoftApp.GetAppDesktop()

Set args = AnsoftScript.arguments
oDesktop.OpenProject dir & CStr(args(0)) & ".hfss"

Set oProject = oDesktop.GetActiveProject()
Set oDesign  = oProject.GetActiveDesign()
Set oModule = oDesign.GetModule("ReportSetup")
```

En la segunda parte del script *proceso.vbs* se crean objetos de los niveles jerárquicos correspondientes y se abre el proyecto identificado por el valor pasado por parámetro y almacenado en la posición 0 del array *args*, array que contiene todos los valores que se pasan como argumentos al script.

Fichero 3.13: proceso.vbs: 3

```
oModule.CreateReport "XY Plot 1", "Modal Solution Data", "Rectangular Plot",
    _
  "Setup1 : Sweep", Array("Domain:=", "Sweep"), Array("Freq:=", Array("All"),
       "a0:=", Array( _
  "Nominal"), "Lport:=", Array("Nominal"), "b0:=", Array("Nominal"), "w1:=",
      Array( _
  "Nominal"), "h1:=", Array("Nominal"), "t1:=", Array("Nominal"), "a1:=",
      Array( _
  "Nominal"), "b1:=", Array("Nominal"), "L1:=", Array("Nominal"), "t2:=",
      Array( _
  "Nominal"), "t3:=", Array("Nominal"), "t4:=", Array("Nominal"), "t5:=",
      Array( _
  "Nominal"), "L2:=", Array("Nominal"), "L3:=", Array("Nominal"), "L4:=",
      Array( _
```

```
    "Nominal"), "x0:=", Array("Nominal"), "w2:=", Array("Nominal"), "w3:=",
        Array( _
    "Nominal"), "h3:=", Array("Nominal"), "w4:=", Array("Nominal"), "h4:=",
        Array( _
    "Nominal"), "h2:=", Array("Nominal"), "w5:=", Array("Nominal"), "h5:=",
        Array( _
    "Nominal"), "a2:=", Array("Nominal"), "a3:=", Array("Nominal"), "a4:=",
        Array( _
    "Nominal"), "b2:=", Array("Nominal"), "b3:=", Array("Nominal"), "b4:=",
        Array( _
    "Nominal"), "Rsop:=", Array("Nominal"), "Hsop:=", Array("Nominal"), "Rres:=
        ", Array( _
    "Nominal"), "Hres:=", Array("Nominal")), Array("X Component:=", "Freq", "Y
        Component:=", Array( _
    "dB(S(1,1))")), Array()
```

En la tercera parte del script *proceso.vbs* se llama a la función *oModule.CreateReport*. Esta función devuelve un report de un proyecto analizado. En su llamada, se asigna un nombre al report, en este caso *XY Plot1*, y se especifica el modo de análisis que fue seleccionado en la fase de preproceso (en este caso un barrido en frecuencia, *Sweep*). Tras ello, establece cuáles son las variables de la estructura e indica el parámetro de análisis a devolver, en este caso, la magnitud en dB del parámetro S_{11}.

Fichero 3.14: proceso.vbs: 4

```
oModule.ExportToFile "XY Plot 1", directorioS & CStr(args(0)) & "S1" & ".csv"
oDesktop.CloseProject CStr(args(0))
oProject.Close
```

Finalmente, se guarda el report en formato *.csv* en el directorio indicado y se cierra el proyecto dejando la interfaz de HFSS cerrada. Se debe comentar que este script debe ser modificado para cada escenario de optimización, diferenciándose, claramente, entre los modos *Sweep* y *No Sweep* de un análisis en HFSS.

Tras obtener los resultados en formato *.csv*, se inicia la segunda fase de postprocesado, fase en la que se lee y manipula la información almacenada y se presenta en un formato visual al usuario. La segunda fase de postprocesado se realiza en Matlab a través de la llamada a la función *controlParalelo.m* en el script principal:

/opt/matlab/bin/matlab -nosplash -nodisplay -nojvm -r controlParalelo

El script *controlParalelo.m* se implementa como sigue:

Fichero 3.15: controlParalelo.m: 1

```
close all;
load workspace
clc;
color = ['b' 'g' 'r' 'c' 'm' 'k'];
lencolor = length(color);
```

```
fre=[];
mag=[];
aleat=[];
index=1;

nomFicheros = dir;
tamano = size(nomFicheros);
indice = 0;
```

En primer lugar, se han definido las variables que se utilizarán a lo largo del script, se han definido las variables *fre* y *mag* para representar frecuencia y magnitud de los parámetros [S] del filtro dieléctrico. Además, se ha definido la variable *color* para yuxtaponer los resultados de las simulaciones realizadas en una misma gráfica con distintos colores.

Fichero 3.16: controlParalelo.m: 2

```
for n = 1:tamano(1,1)
nombre= nomFicheros(n,:).name;
    if isequal(nombre(1,1:1),'p')
        if isequal (nombre(1,1:2),'pa')
            if isequal (nombre(1,1:4),'para')
                if isequal (nombre(1,1:6),'paramS')
    indice = indice + 1;
    numero='';
    nom = nomFicheros(n,:);
        s=7;
        if ( strcmp(nombre(1,s+2),'c') == 1 ) && ( strcmp(nombre(1,s+3),'s')
            == 1 ) && (strcmp(nombre(1,s+4),'v') == 1) && (strcmp(nombre(1,s
            +1),'.') == 1)
        numero =  nombre(1,s);
        end
```

El comando *dir* se ha utilizado para listar el contenido del directorio en el que nos encontramos. Una vez almacenado dicho directorio, se recorren todos los ficheros casi a nivel de caracter a caracter, para encontrar los resultados en formato *.csv* devueltos por HFSS como resultado del análisis de cada subproyecto. Nótese que realizamos la búsqueda casi caracter a caracter para no acceder a posiciones ilegales. Esto ocurre por la posible existencia de ficheros, cuyos nombres, tienen una longitud menor en caracteres que la cadena con la que comparamos. Por último, revisamos que efectivamente se trata de un archivo de extensión *.csv*.

Fichero 3.17: controlParalelo.m: 3

```
    [fid mensaje] = fopen(nom.name);
    A = fscanf(fid, '%g');
    cell = textscan(fid, '%s');
    arrayS = cell{1};
    datosS = arrayS(4:length(arrayS));
    M = zeros(length(datosS),1);
    secuence = 1:length(datosS);

        for i=1:length(datosS)
           m = datosS{[i]};
```

```
                fout = fopen('salida.csv','a+');
12              fprintf(fout,'%s',m);
                fprintf(fout, '\n');
                fout2 = fopen('ordenados.csv','a+');
                fprintf(fout2,'%s',m);
                fprintf(fout2, '\n');
17              fclose(fout2);
                fclose(fout);
            end

        fclose(fid);
```

Para el tratamiento de la información que nos reporta HFSS se debe hacer una gestión de ficheros, en primer lugar, abriendo el fichero de salida de HFSS.

Cuadro 3.1: Report de HFSS en formato *.csv*

Freq [GHz]	"dB(S(1,1)) [ ]"
8.5	-1.34685468302328
8.6	-1.33339180127875

Si se observa el formato del fichero de salida se puede ver que es un array de celdas, donde la primera celda nos da información del parámetro simulado en la estructura, en este caso, la magnitud en dB del parámetro S_{11} con respecto a la frecuencia. En las sucesivas celdas tenemos la frecuencia concatenada con el caracter ',' y concatenado con la magnitud en dB para ese punto de frecuencia en concreto. Es por tanto que deberemos suprimir la primera celda y separar frecuencia y magnitud en dos arrays distintos para una manipulación correcta de la información. Con $cell = textscan(fid, \%s)$ se lee un fichero, volcándolo en un cell array, estructura que permite construir matrices en las que cada fila es diferente. Una vez se ha obtenido dicho array, se suprime la primera fila de información de análisis y se vuelca la información restante en un fichero de salida que denominaremos *salida.csv*.

El fichero *salida.csv* será borrado entre report y report para evitar que se apilen en él los resultados. Por esta razón, también es necesario el fichero *ordenados.csv*, cuyo contenido será precisamente la apilación de los resultados de todos los reports.

El fichero *salida.csv* nos permitirá ir representando sucesivamente y de forma yuxtapuesta los resultados de los reports en un gráfica, mientras que, el fichero *ordenados.csv* se utilizar en la función de evaluación del algoritmo genético.

Fichero 3.18: controlParalelo.m: 4

```
load salida.csv
frecuencias = salida(:,1);
magnitud = salida(:,2);

fre=[];
mag=[];
aleat=[];
    fre(indice,:) = frecuencias;
    mag(indice,:) = magnitud;
    aleat = round(lencolor*(rand(1,1)));
        if aleat == 0
        aleat = 1;
        end
aleat;
color(aleat);
hold on

fre(index,:);
mag(index,:);
color(aleat);
handle = plot(fre(index,:),mag(index,:),color(aleat));
index=index+1;
    end
        end
        end
        end
delete salida.csv
end
```

Una vez se ha obtenido el fichero en el formato deseado, podemos acceder a la frecuencia y magnitud a nuestro placer y representar ,con la función *plot*, magnitud Vs frecuencia. Nótese que almacenamos el manejador de la figura para su posterior almacenamiento.

Fichero 3.19: controlParalelo.m: 5

```
title('param [S] (db) vs frecuencia');
save figure
num2str(numero);
saveas(handle(length(handle)),'resultados.eps','psc2');
save matlab.mat

load ordenados.csv
sest = ordenados(:,2);
delete ordenados.csv;
fout2 = [];

save workspace
exit
```

Para salvar la figura guardamos el valor del manejador que devuelve la función plot y usamos la función *saveas* para almacenar la figura en formato *eps*, formato que nos permitirá aumentar la resolución de la figura sin perder precisión.

Además, se guardan en la variable *sest* todos los resultados de todos los reports para su manipulación en la función *evaluación* del algoritmo genético.

Si se aplica una simulación paramétrica distribuida sobre el filtro dieléctrico presentado a modo de ejemplo, se obtienen los resultados de la figura 3.7.

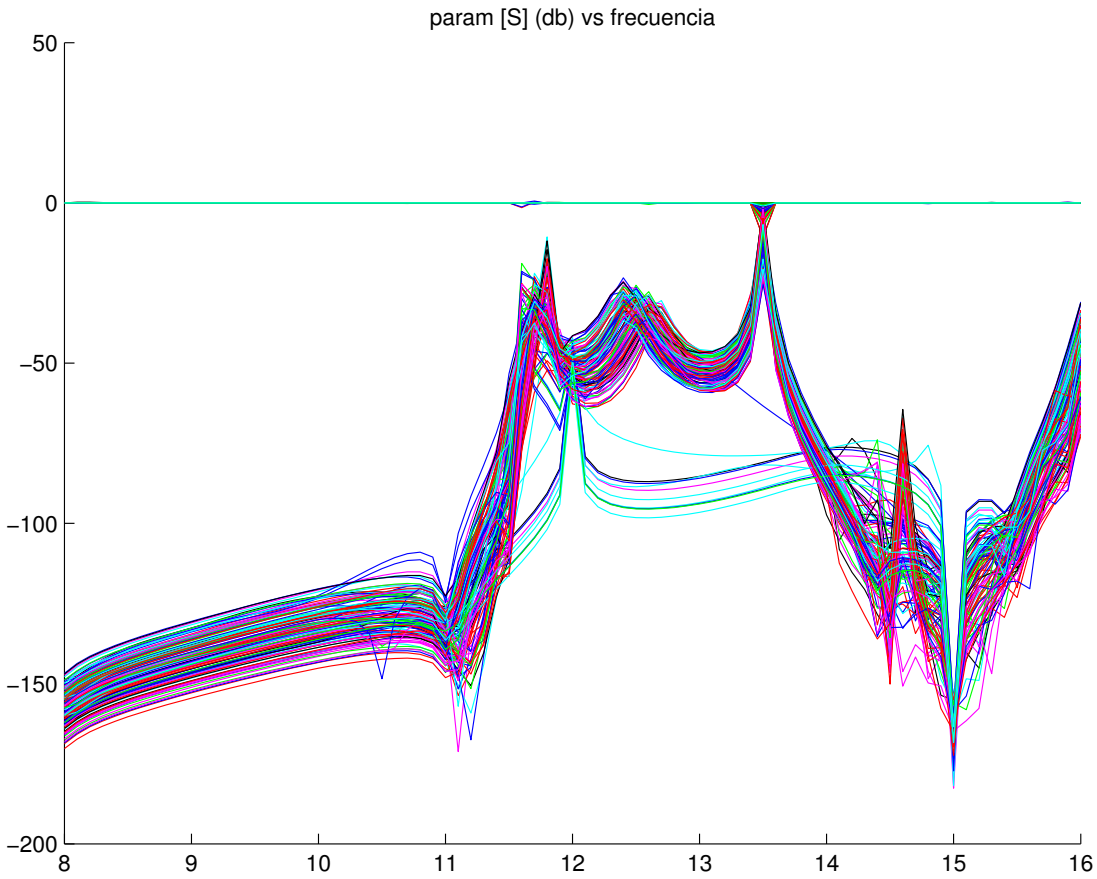


Figura 3.7: Estudio de sensibilidad del filtro dieléctrico

La resolución secuencial de estas 125 simulaciones, duraría aproximadamente, 49 horas en un ordenador empleando un sólo núcleo, doce horas y media en un Intel Core 2 Quad, mientras que, empleando veinte nodos del cluster, con ocho núcleos cada uno, dura 32 minutos.

Capítulo 4

Herramienta de optimización

Debido a la complejidad existente en el diseño de ciertas estructuras electromagnéticas (como se vió en el capítulo 1), el diseño de las mismas se convierte en una tarea ardua debido al elevado número de parámetros que presentan. La optimización es por tanto, un proceso necesario en el diseño de una estructura de análisis complejo.

Optimizar es, en definitiva, buscar de entre todas las alternativas posibles, aquellas que mejoran el resultado para un problema específico.

En el proceso de optimización se necesitan:

- Los datos deseados $d(k)$. Valores ideales teóricos de los parámetros que se desean analizar.
- Datos que proporciona nuestro sistema, $o^{(k)} = F_w(x^{(k)})$, acumulando sus valores según k. Son los valores de los parámetros de análisis obtenidos tras la simulación.

Con los datos anteriores se obtiene una curva de coste (Figura 4.1) a minimizar, cuya variación con w es arbitraria y donde w depende de los datos obtenidos por nuestro sistema. La curva de coste C será una relación entre los datos deseados y los datos obtenidos tras la simulación.

La Figura 4.1 dispone de dos mínimos, uno local y otro global, los algoritmos de búsqueda se clasifican del mismo modo en locales y globales.

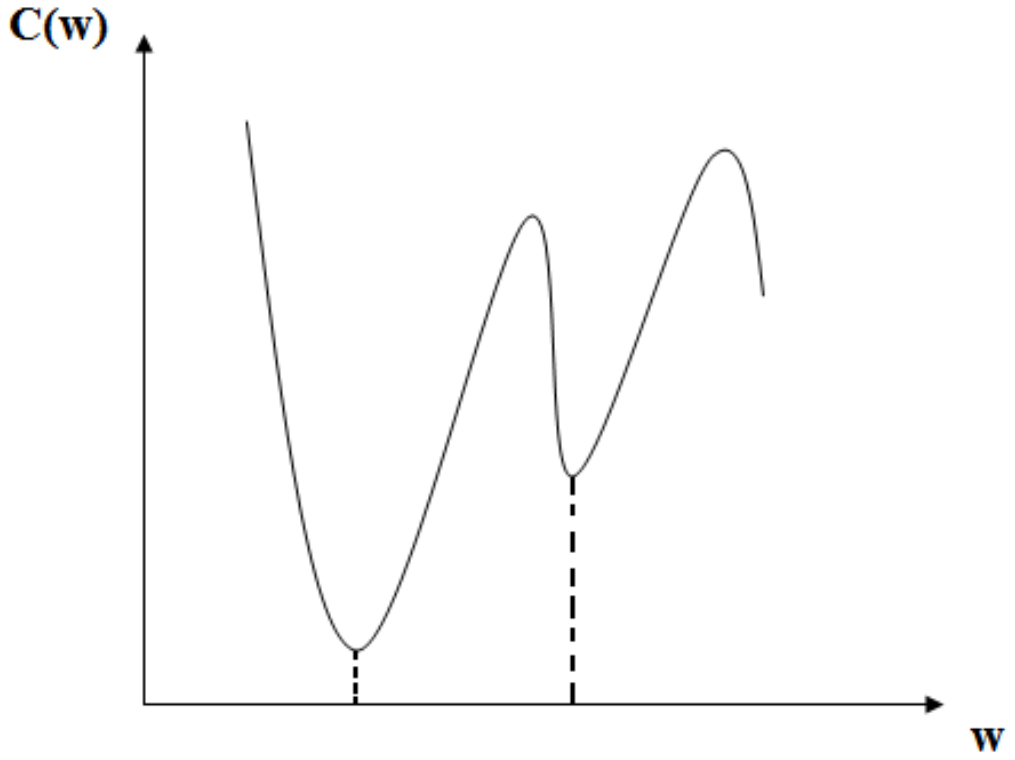


Figura 4.1: Función de coste en función de w.

Los algoritmos de búsqueda locales pueden encontrar un mínimo relativo a partir del punto en el que se inicializa el algoritmo. De esta forma, el mínimo de la función de coste, se localiza en la cuenca de atracción del punto inicial.

En este punto se puede deducir que, un algoritmo de búsqueda local no satisface nuestras necesidades, pues necesitamos encontrar de entre las posibles alternativas, la mejor. Debido a que se desea encontrar el mejor valor o valores de los parámetros que componen la estructura, se ha seleccionado la búsqueda global.

Téngase en cuenta que se podría seguir la estrategia de inicializar, en diferentes puntos de la función de coste, un algoritmo de búsqueda local y seleccionar el mejor mínimo encontrado. Sin embargo, el optimizador sería subóptimo y no resolvería el problema de optimización convenientemente. Sin contar que, la estructura bajo análisis está indeterminada y que los resultados que podemos extraer de su análisis pueden ser muy diversos y arrojar funciones matemáticas complejas.

Existen varios métodos de búsqueda global [13], algunos ejemplos son:

- Recocido simulado (Simulated Annealing)

- Búsqueda tabú.
- GRASP (Greedy Randomized Adaptive Search Procedure)
- Algoritmo Genético.
- Algoritmo de colonia de hormigas.

Se debe analizar cuál es la opción que mejor satisface nuestras necesidades de entre los posibles algoritmos de búsqueda conocidos. Una posible solución es el *recocido simulado* o *simulated annealing*, que se basa en el proceso industrial en el que un material se calienta por el punto de fusión y luego se enfría gradualmente para eliminar defectos en su estructura cristalina, produciendo una estructura más estable y regular. A diferencia del algoritmo genético que posee una población de soluciones candidatas, este algoritmo solo cuenta con una solución candidata.

Simulated annealing añade el concepto y valor de temperatura, cantidad número-rica que disminuye gradualmente con el tiempo hasta que alcanza el valor cero y se congela, obteniéndose así, la solución final. El algoritmo dispone también de una medida de aptitud, comparando aptitudes entre medidas consecutivas, y seleccionando la mejor de ellas.

Una mala elección del punto inicial o del esquema de enfriamento (actualización de la temperatura) puede implicar la incorrecta exploración del espacio de búsqueda.

Este método funciona en serie por lo que se explora el espacio de búsqueda hacia una solución en una dirección al mismo tiempo, debido a esto, no seleccionaremos *simulated annealing* [13] como algoritmo de búsqueda en nuestro sistema, sino que nos decantaremos por los algoritmos bioinsipirados que sí permiten la paralelización y permiten, en el peor de los casos, un ataque de fuerza bruta sobre todo el espacio de búsqueda, consiguiéndose, sin discusión, la solución óptima.

El primero de los algoritmos bioinspirados [14] bajo estudio, parte del comportamiento de las hormigas. Estos vienen a entender la colonia como un único organismo cuyos órganos son dichas hormigas. La comunicación entre las hormigas es limitada, de forma que esta se lleva a cabo a través de los actos realizados anteriormente por otros miembros de la colonia, o a través de la acumulación de feromonas en el suelo.

El algoritmo consiste en tomar un conjunto de hormigas y soltarlas en un grafo, donde los puntos representan ciudades a visitar (soluciones candidatas). Las hormigas colocarán en cada arco del grafo (unión entre dos ciudades), una cantidad de feromonas inversamente proporcional a la distancia que une esas dos ciudades.

La hormiga elegirá, con mayor probabilidad, el camino que posea una mayor cantidad de feromonas, haciéndose cada vez más probable, el camino mas corto. Tras el paso del tiempo todas las hormigas encontrarán la solución óptima.

El inconveniente de esta solución es que todas las hormigas de la colonia deben encontrar la solución óptima, mientras que, el algoritmo genético puede ser parardo en cualquier momento para realizar la evaluación de la solución escogida.

Debido a los inconvenientes presentados en las anteriores técnicas de búsqueda y por las ventajas que se presentarán a continuación, hemos optado por seleccionar el algoritmo genético [15] como algoritmo de búsqueda en nuestro sistema.

Siguiendo la definición dada por Goldberg, *"los Algoritmos Genéticos son algoritmos de búsqueda basados en la mecánica de selección natural y de la genética natural. Combinan la supervivencia del más apto entre estructuras de secuencias con un intercambio de información estructurado, aunque aleatorizado, para constituir así, un algoritmo de búsqueda que tenga algo de la genialidad de la búsqueda humana"* [16].

Para alcanzar la solución a un problema se parte de un conjunto inicial de individuos llamado población, generado de manera aleatoria. Cada uno de estos individuos representa una posible solución al problema.

La población existente generará descendencia de acuerdo a los mecanismos reproductivos que tienen en cuenta los fenómenos de cruce y mutación. Las poblaciones posteriores se adaptarán al medio mediante la selección de los invidivuos más aptos.

Este algoritmo presenta un conjunto de ventajas que fundamentan su elección frente a las técnicas vistas:

1. El primer y más importante punto es que los algoritmos genéticos son intrínsecamente paralelos. Las técnicas vistas son en serie y sólo pueden explorar el espacio de soluciones hacia una solución en una dirección al mismo tiempo. Si la solución que descubren resulta subóptima, se debe empezar de nuevo; sin embargo, los algoritmos genéticos tienen descendencia múltiple, pueden explorar el espacio de soluciones en múltiples direcciones a la vez dando una mayor probabilidad en cada ejecución al camino óptimo.

2. Al evaluar la aptitud de un individuo concreto se evalúan los espacios a los que pertenece. Dada esta característica, los algoritmos genéticos funcionan particularmente bien resolviendo problemas cuyo espacio de soluciones es muy grande. Problemas que pueden ser lineales o no.

3. Otra ventaja de los algoritmos genéticos es que se desenvuelven bien en problemas en los que la función de aptitud es discontinua, ruidosa, cambia con el tiempo, o tiene muchos óptimos locales. La mayoría de los problemas prácticos que nos encontraremos tienen un espacio de soluciones enorme, lo que propicia el uso de este algoritmo.

4.1. Algoritmo genético

Como se indicó en la introducción de este capítulo, para encontrar la solución a un problema se parte de un conjunto inicial de individuos, llamado población, generado de manera aleatoria. Cada uno de estos individuos representa una posible solución al problema. Estos individuos evolucionarán adaptándose al medio y generarán poblaciones descendientes mediante su reproducción, mecanismo evolutivo caracterizado por el cruce genético y la mutación.

Los algoritmos evolutivos son como sabemos especialmente útiles cuando nos encontramos con problemas de elevada complejidad, como lo son los caracterizados por una alta dimensionalidad, fuerte no linealidad, presencia de ruido, etcétera. Trabajan con una población aleatoria de individuos, que representan soluciones candidatas a un problema. Esta población se somete a ciertas transformaciones y después a un proceso de selección, que favorece a los individuos más aptos.

En cada iteración del proceso evolutivo se van seleccionado los individuos con mejor aptitud, de forma que se tiende a la solución óptima progresivamente. Los algoritmos evolutivos combinan la búsqueda aleatoria, dada por las transformaciones de la población, con una búsqueda dirigida dada por la selección.

El algoritmo genético ha sido implementado en Matlab por la gran capacidad de este en el tratamiento de datos. Dispone de las siguientes etapas:

- Inicialización.
- Cruce.
- Mutación.
- Reproducción.
- Evaluación.

En la Figura 4.2 se muestra el orden de los operadores del algoritmo genético integrado en el sistema completo, donde la caja rellena de color amarillo es la simulación paramétrica distribuida descrita en el capítulo 3.

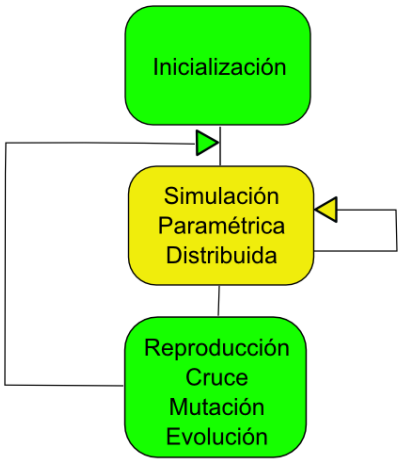


Figura 4.2: Diagrama de flujo sistema completo

4.1.1. Inicialización

El primer paso del algoritmo genético es la inicialización. En ella se parte de un conjunto inicial de individuos que se denomina población, y que es generado aleatoriamente.

Cada individuo representa una posible solución del problema, donde la solución final es el valor o valores del parámetro o parámetros que optimizan la estructura. La codificación del individuo considera pues, que el individuo está formado por uno o más parámetros, donde cada parámetro o gen se codifica con una cadena binaria de una longitud determinada. La concatenación de los genes componen el individuo, que se caracteriza por este cromosoma.

El grado de precisión del algoritmo es muy flexible, ya que este se ha diseñado de forma que cada parámetro pueda tener una resolución distinta.

En definitiva, la población es un conjunto de individuos, donde cada individuo se codifica por una etiqueta y un conjunto de genes (cromosoma), y donde la etiqueta se usa para no perder la localización del individuo dentro de la población tras el cruce, mutación y reprodución.

Cada gen se codifica de manera independiente con un número de bits que determinan la precisión del mismo, por tanto, se necesita establecer para cada parámetro, un rango de valores sobre el que seleccionaremos $2^{numerodebits}$ posibles soluciones, donde obtendremos, de entre estas posibles soluciones, el valor que optimiza la esctructura.

A continuación, se detalla la inicialización empleada para un filtro dieléctrico de 4 cavidades, donde utilizaremos 3 parámetros para la optimización.

Fichero 4.1: inicializacion.m: Inicialización

```
clc;
clear all;
close all;

nbits=4;
mbits=3;
lbits=5;
totalbits=nbits+mbits+lbits;

pE = 0.8;
pC = 0.8;
pM = 0.01;

poblacion = round(rand(2^nbits,totalbits));

id=1;
ident = [num2str(id) '.hfss'];
individuo(2^nbits) = struct('ident','','valor','');
        for n=1:2^nbits
        individuo(n)= generacion(poblacion(n,:),id);
        id=id+1;
        end
gen1 = struct('variable','','ident','','unidad','','centro','','spam','','
    nbits','');
gen1.variable = 'Hres';
gen1.ident = '1';
gen1.unidad= 'mm';
gen1.centro=2.3;
gen1.spam=0.1;
gen1.nbits=nbits;

%Definimos diversos genes que componen el cromosoma

genes = struct('ident','','valor','');
genes(1).ident = 'gen1';
genes(1).valor = gen1;
genes(2).ident = 'gen2';
genes(2).valor = gen2;
genes(3).ident = 'gen3';
genes(3).valor = gen3;

%Insertamos en el array de genes todos los genes definidos

save workspace

disp('fin de inicializaci n')
exit
exit
```

Tras resetear el directorio de trabajo, el diseñador establece qué parámetros va a optimizar y la resolución de cada parámetro (*nbits,mbits y lbits*).

Se establecen las probabilidades de cruce, mutación y reproducción y se inicializa una población aleatoria con un número determinado de individuos, en el que cada individuo resulta de la concatenación de los genes que lo componen, y donde

cada gen o parámetro, es una secuencia binaria que codifica un posible valor del parámetro. Para finalizar, se inicializa *individuo*, estructura de matlab (*struct*) que representa un array de individuos etiquetados.

La segunda parte de la inicialización consiste en definir los genes que componen el cromosoma que caracteriza a un individuo. De esta forma, se define un parámetro o gen por: un identificador, por la resolución determinada para el mismo, por las unidades de medida usadas para caracterizar el valor adquirido y por el punto inicial y paso entre valor y valor, mediante los cuales decodificamos la secuencia binaria al valor real que adquiere el parámetro.

Con lo anterior, se inicializa un array de *struct*, donde cada posición del array es cada gen que compone el cromosoma. Por último, se guarda el directorio de trabajo para salir de matlab (*exit*) y no perder el valor de las variables declaradas.

La codificación diseñada consiste por tanto, en coger un rango de valores y seleccionar aleatoriamente puntos de ese rango, de forma que se fija la precisión del algoritmo con el número de puntos escogidos. El paso entre un valor del parámetro y el siguiente está fijado por la resolución: a mayor número de bits utilizados para codificar el valor del parámetro, mayor resolución. El usuario se acercará con mayor precisión a la solución, aumentando el número de bits y centrando el rango de valores escogido en la solución óptima.

Tras el análisis anterior se puede modificar el código para adaptarlo a las necesidades que se deducen de cada escenario de optimización.

4.1.2. Cruce

El operador de cruce [17] [15] genera una descendencia con el mismo número de individuos que la población de la que procede. Se trata por tanto, de una nueva población de individuos cuyo cromosoma genético es el producto de la recombinación del de sus padres. Se ha optado por realizar una operación de cruce por un punto, la cual consiste en, una vez seleccionados dos individuos, cortar sus cromosomas por un punto y recombinar las cadenas genéticas generadas tras el corte. Es destacable el hecho de que la operación de cruce puede producirse o no, es por ello que modelamos este suceso con una probabilidad P_c, produciéndose así, el cruce genético en el caso de que la hipótesis de cruce se cumpla, y la misma población en el caso de que no se cumpla.

Cruce y Mutación se utilizan para explorar el espacio de búsqueda. Aumentando la probabilidades de mutación y cruce, o variando el número de puntos de corte o

número de bits susceptibles de ser mutados, se modifica la potencia del algoritmo para realizar búsquedas más o menos precisas sobre el espacio de búsqueda.

Para el dispositivo bajo estudio y dado que tenemos conocimiento teórico de la solución, no resulta interesante localizar dicha solución en el espacio de búsqueda analizando puntos muy dispersos, es por eso, que hemos seleccionado cruce por un punto.

Se ilustra este operador con el siguiente ejemplo: pensemos en dos individuos de la población, codificados en 11 bits.

- Individuo 1 (I1) : 10110000111
- Individuo 2 (I2) : 11110101001

El punto de corte para realizar la operación de cruce ha sido seleccionado aleatoriamente, en este caso ha sido seleccionada la posición 6 de la cadena binaria, ver tabla 4.1.

Cuadro 4.1: Posición y valor binario

Posición	1	2	3	4	5	6	7	8	9	10	11
Valor binario	1	0	1	1	0	0	0	0	1	1	1

Tras realizar el corte se obtienen las secuencias genéticas de las tablas 4.2 y 4.3.

Cuadro 4.2: Individuo 1 y secuencias de corte

Individuo 1 (I1)	Secuencia de corte S11	Secuencia de corte S12
10110000111	101100	00111

Cuadro 4.3: Individuo 2 y secuencias de corte

Individuo 2 (I2)	Secuencia de corte S21	Secuencia de corte S22
11110101001	111101	01001

Tras recombinar las secuencias genéticas, obtenemos como resultado dos individuos de secuencias [S11 S22] y [S12 S21]. Ver tabla 4.4.

La función *cruce.m* implementa esta funcionalidad como se detalla a continuación:

Cuadro 4.4: Individuos tras el operador corte

Individuo resultante 1	Individuo resultante 2
[S11 S22] 10110001001	[S12 S21] 11110100111

Fichero 4.2: recrumu.m: Cruce

```
function cruzados = cruce(poblacion,pC)

long = length(poblacion(1).valor);
mitad = long/2;
corte = ceil(mitad);


N = length(poblacion);
M = length(poblacion(1).valor);
aleatC = rand(1);

cruzados(length(poblacion)) = struct('ident','','valor','');

t=1;
    while t <= N
    b1 = poblacion(t).valor;
    b2 = poblacion(t+1).valor;

    S11 = b1(1,1:corte);
    S12 = b1(1,corte+1:M);
    S21 = b2(1,1:corte);
    S22 = b2(1,corte+1:M);

            if aleatC > 1-pC
            hijo1 = [S11 S22];
            hijo2 = [S21 S12];
            cruzados(t)=struct('ident',poblacion(t).ident,'valor',hijo1);
            cruzados(t+1)=struct('ident',poblacion(t+1).ident,'valor',hijo2);
            else
            cruzados(t)=poblacion(t);
            cruzados(t+1)=poblacion(t+1);
            end
    t= t+2;
    end
```

La función *cruce.m* recibe como argumentos una población y una probabilidad de cruce. Esta función establece en primer lugar, el punto de corte. Este punto es una posición aleatoria que ha sido fijada al punto medio, pero que podría ser perfectamente otra. Después, se inicializa *cruzados*, que se trata de un *struct* que contiene la población tras el operador de cruce y que es devuelto como salida por nuestra función.

Tras lo anterior, se obtienen los cromosomas de dos individuos consecutivos (*b1 y b2*) y se obtienen las secuencias binarias resultantes tras realizar el corte, tal y como se ilustra en la tabla 4.2 y 4.3.

Con las secuencias de corte se obtienen los individuos cruzados, siempre y cuando, se cumpla la hipótesis de superar la probabilidad de cruce. De esta forma, se almacena en la variable *cruzados* la nueva población, compuesta de los individuos cruzados con probabilidad P_c y los mismos individuos con probabilidad $1 - P_c$.

4.1.3. Mutación

La mutación [17] de un individuo provoca que alguno de sus genes varíe su valor, por lo tanto, cada individuo generado por el operador de cruce será susceptible de ser afectado por la mutación.

Una vez seleccionado el individuo *i-esimo* de la población, se selecciona aleatoriamente la posición de bit en su secuencia genética y se altera el valor binario correspondiente a esa posición con probabilidad P_m. La probabilidad de mutación P_m es normalmente baja (menor al 1 %), ya que los individuos mutados suelen tener un fitness menor después de la mutación. Sin embargo, esta operación es importante para asegurarnos de que ningún punto del espacio de búsqueda quede sin ser explorado.

Fichero 4.3: recrumu.m: Mutación

```
function mutado = mutacion(cruzados,pM)
N = length(cruzados);
M = 4;
aleatM = rand(1);
mutado(length(cruzados)) = struct('ident','','valor','');
        for l=1:N
        aleatM = rand(1);
                if aleatM > 1-pM
                bitposicion = round(M*rand(1));
                        if bitposicion == 0
                        bitposicion = 1;
                        end
                temp= cruzados(l).valor;
                valorC = temp(bitposicion);
                valorE =str2num(valorC);
                noValorE = not(valorE);
                noValorC = num2str(noValorE);
                temp(bitposicion)= noValorC;
                mutado(l).valor = temp
                mutado(l).ident = cruzados(l).ident;
                else
                mutado(l).valor = cruzados(l).valor;
                mutado(l).ident = cruzados(l).ident;
                end
        end
```

El fichero 4.3, recibe como parámetros una población y una probabilidad P_m, y devuelve la población resultante tras realizarse la operación de mutación.

Como puede verse en el código anterior, se obtiene con probabilidad P_m una posición aleatoria dentro de la cadena genética mediante *bitposicion = round(M*rand(1));*. A continuación, se obtiene el cromosoma del invididuo y se almacena el valor del cromosoma evaluado en la posición *bitposicion*.

Una vez obtenido este valor (de tipo *char*), se convierte a entero con la función *str2num*, se niega su valor, se reconvierte a una variable de tipo *char* con la función *num2str* y se introduce mutado en la cadena genética.

Es importante notar que el individuo se conserva en la nueva población con probabilidad $1 - P_m$.

4.1.4. Reproducción

Cada individuo presenta una medida de fitnes que mide su capacidad o respuesta ante una determinada utilidad, siendo el individuo con mas fitnes el invidivudio que presenta mayor capacidad ante la utilidad. Como ejemplo, considérese el escenario de una carrera donde medimos como utilidad la rapidez del individuo. En este caso, el que presenta una mayor capacidad, una mayor rapidez, es aquel individuo que llegue antes a la meta. Si medimos el tiempo que necesita cada invidivuo para llegar a la meta estaremos midiendo el fitnes de ese individuo con respecto a su capacidad por llegar antes a la meta.

De este modo y mediante el operador de selección conseguiremos que los individuos de la siguiente población sean los de mayor fitnes, los más aptos, modelándose, de esta forma, la selección natural.

La selección de candidatos para la reproducción es un punto importante en un algoritmo genético y existen diversas opciones: selección proporcional, selección mediante torneo y selección de estado uniforme.

En nuestro caso nos hemos decantado por usar una estrategia de selección mediante torneo [18]. Estrategia que realiza la selección en base a la comparación directa de sus individuos. Esta estrategia consigue valorar más a los mejores individuos y centrar la búsqueda de la solución en los individuos aptos.

La selección mediante torneo se presenta en dos versiones: determinista y probabilística.

La versión determinista garantiza que el mejor individuo será seleccionado n veces, con n el tamaño del torneo. Sin embargo, presenta como inconveniente que puede

introducir una presión selectiva muy alta (los individuos con bajo fitness no tiene probabilidad de sobrevivir).

Por ello, nos decantamos por la versión probabilística, idéntica a la versión determinista salvo por la elección del ganador del sorteo. Mientras que, en la versión determinista seleccionamos siempre al individuo con mayor fitness, en la versión probabilista seleccionamos al individuo de mayor fitness con una probabilidad p.

Pongamos como ejemplo una población de 6 individuos cada uno con un fitness asociado (tabla 4.5), y considerándose la probabilidad $p = 1$, método determinista.

Cuadro 4.5: Individuo Vs Fitness

Individuo	Fitness
1	23
2	54
3	67
4	10
5	20
6	88

A continuación, se barajan los individuos n veces para asegurar que el mejor individuo es seleccionado n veces. Las barajas son aleatorias. Ver tabla 4.6.

Cuadro 4.6: Resultado de las barajas

Baraja 1	Baraja 2
6	5
3	1
5	2
1	3
2	4
4	6

Para cada baraja enfrentamos los individuos dos a dos comparando su fitness y seleccionando al de mayor fitness con probabilidad p. Los resultados del torneo para las barajas dadas pueden verse en las tablas 4.7 y 4.8.

Cuadro 4.7: Ganadores del torneo en la baraja 1

Baraja 1 - Individuo (fitness)	Ganadores baraja 1 - Individuo
6(88)	6(88)
3(67)	
5(20)	5(20)
1(23)	
2(54)	2(54)
4(10)	

Cuadro 4.8: Ganadores del torneo en la baraja 2

Baraja 2 - Individuo (fitness)	Ganadores baraja 2 - Individuo
5(20)	5(20)
1(23)	
2(54)	3(67)
3(67)	
4(10)	6(88)
6(88)	

Una vez seleccionados los vencedores de ambas barajas se obtiene la siguiente población superviviente (tabla 4.9), que como se puede ver, obtiene un fitness global mayor al de la población inicial.

Esta funcionalidad ha sido implementada en la función *reproduccion.m*, descrita a continuación:

Fichero 4.4: recrumu.m: Reproducción

```
function gana = reproduccion(barajo1,barajo2,coste,pE)
coste1 =coste(barajo1);
coste2 =coste(barajo2);
N = length(coste);
gana(length(coste))= struct('ident','','valor','');
y=1;
m=1;
    while m < N
    aleatPE = rand(1);
        if aleatPE<pE
            if (coste1(m).valor < coste1(m+1).valor)
             gana(y).valor=coste1(m).individuo;
             gana(y).ident=coste1(m).ident;
            else
             gana(y).valor=coste1(m+1).individuo;
             gana(y).ident=coste1(m+1).ident;
            end
        else
```

```
            gana(y).valor=coste1(m).individuo;
            gana(y).ident=coste1(m).ident;
            end
        y=y+1;
        m=m+2;
        end
m=1;

        while m < N
        aleatPE = rand(1);
            if aleatPE<pE
                if (coste2(m).valor< coste2(m+1).valor)
                gana(y).valor=coste2(m).individuo;
                gana(y).ident=coste2(m).ident;
                else
                gana(y).valor=coste2(m+1).individuo;
                gana(y).ident=coste2(m+1).ident;
                end
            else
            gana(y).valor=coste1(m).individuo;
            gana(y).ident=coste1(m).ident;
            end
        y=y+1;
        m=m+2;
        end
```

Cuadro 4.9: Individuos supervivientes

Individuo Superviviente
6
5
2
5
3
6

La función *reproduccion.m* recibe como parámetros 2 barajas, el fitness asociado a cada individuo y la probabilidad de que suceda la reproducción P_e. Devuelve como salida la población superviviente tras la selección modelada por torneo en versión probabilística.

Con probabilidad P_e se produce la reproducción. Una vez que nos encontramos en esta hipótesis, seleccionamos con probabilidad p el individuo que minimiza la función de coste y con probabilidad $1-p$ el individuo contrario.

Con probabilidad $1-P_e$ se conservan los mismos individuos sin aplicar la selección por torneo.

4.1.5. Evaluación

El fitness con el que se ha resuelto el mecanismo de selección anterior debe ser asignado a cada individuo. En el caso de estudio, el algoritmo de búsqueda trata de encontrar el mínimo global de una función de coste arbitraria, es por este motivo, por el que se fijará como fitness de cada individuo el coste obtenido de evaluar la función de coste en el individuo *i-esimo*.

En nuestro particular minimizaremos el error cuadrático medio definido por:

$$C(s, \hat{s}) = \sum w_i (s_i - \hat{s_i})^2$$

Donde s son los valores deseados de la función a minimizar y $\hat{s}$ los estimados (obtenidos por nuestro sistema).

4.2. Transformador $\lambda/4$

Para ilustrar el optimizador diseñado se ha seleccionado como estructura a optimizar un transformador $\lambda/4$, estructura ampliamente conocida y cuyo resultado conocemos teóricamente a priori. Además, se ha validado dicho resultado con el sotware MWO [19].

Esta estructura ha sido seleccionada por su simplicidad, ya que no es objeto de este estudio profundizar en el análisis y resolución de estructuras de microondas, sino validar con ellas una metodología. Aún así, es necesario al menos, resolver teóricamente este caso para entender físicamente lo que se deduce de un análisis tipo.

Es común que se deba conectar una carga a una línea de impedancia característica diferente. En tal caso, existirá una onda reflejada que disminuye la potencia entregada a la carga y puede tener efectos adversos en el generador, crear sobretensiones y sobrecorrientes sobre la línea capaces de causar daños, etc. Para evitar estas situaciones problemáticas existen distintos mecanismos de adaptación entre la línea y la carga, en la Figura 4.3 se detalla el transformador $\lambda/4$.

Se trata de una sección de línea de longitud L y de impedancia característica Z_1. Para la adaptación, se requiere que la impedancia de entrada del conjunto carga y adaptador sea igual a la impedancia característica de la línea original Z_2.

El parámetro a optimizar en la línea de transmisión es la longitud eléctrica L. Nuestro objetivo será obtener el valor óptimo, de forma que los parámetros [S] calculados, sean lo mas parecido posible a los resultados analíticos. A continuación, se muestra el análisis teórico del transformador $\lambda/4$, obteniéndose de este análisis los parámetros [S].

Observando la estructura, se puede determinar a priori, que la red es físicamente simétrica y recíproca por lo que $S_{11} = S_{22}$ y $S_{21} = S_{12}$. De esta forma, se puede caracterizar la matriz de parámetros [S] obteniendo dos de sus parámetros, el S_{11} y S_{21}.

$$S_{11} = \frac{b_1}{a_1} \quad \forall a_2 = 0 \qquad\qquad S_{21} = \frac{b_2}{a_1} \quad \forall a_2 = 0$$

Primero se calculan las tensiones y corrientes en los puertos, para obtener con ellas las ondas de potencia. Considerando el sentido positivo hacia la carga y con el origen en esta, la tensión y la corriente en cada punto de la línea de transmisión viene dada por las expresiones:

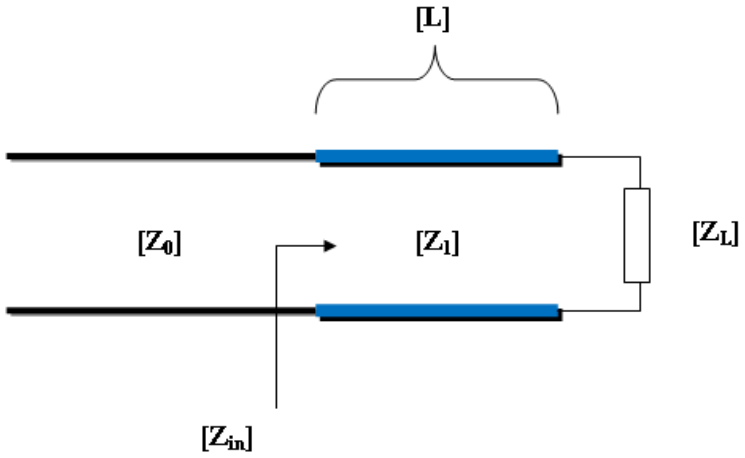


Figura 4.3: Linea de transmisión.

$$V(z) = V_0^+(e^{-j\beta z} + \Gamma_L e^{j\beta z})$$

$$I(z) = \frac{V_0^+}{Z_1}(e^{-j\beta z} - \Gamma_L e^{j\beta z})$$

Donde V_0^+ es la amplitud de la onda progresiva (en sentido generador a carga) en el origen de coordenadas (la carga en este caso) y Γ_L es el coeficiente de reflexión de la carga Z_0 con respecto a la impedancia característica de la línea Z_1.

$$\Gamma_L = \frac{Z_0 - Z_1}{Z_0 + Z_1}$$

Nótese que no hace falta determinar el valor de V_0^+ dado que se cancelará al calcular los parámetros [S] cuando se comparen las ondas de potencia. Tomando $z = 0$ se obtiene la tensión y la corriente en la carga.

$$V_2 = V(0) = V_0^+(1 + \Gamma_L)$$

$$I_2 = -I(0) = -\frac{V_0^+}{Z_1}(1 - \Gamma_L)$$

Por su parte con $z = -\lambda/4$ se obtiene la tensión y corriente a la entrada de las sección de la línea de transmisión.

$$V_1 = V(-\lambda/4) = V_0^+(e^{j\pi/2} - \Gamma_L e^{-j\pi/2}) = jV_0^+(1 - \Gamma_L)$$

$$I_1 = I(-\lambda/4) = \frac{V_0^+}{Z_1}(e^{j\pi/2} - \Gamma_L e^{-j\pi/2}) = j\frac{V_0^+}{Z_1}(1 - \Gamma_L)$$

Aplicando directamente su definición, se pueden calcular las ondas de potencia. Téngase en cuenta que $V_2 = -Z_0 I_2 (a_2 = 0)$ y que la impedancia de referencia para definir los parámetros [S] es Z_0.

$$a_1 = \frac{V_1 + Z_0 I_1}{\sqrt{8}\, Z_0} = \frac{jV_0^+}{\sqrt{8Z_0}}[(1 - \Gamma_L) + \frac{Z_0}{Z_1}(1 + \Gamma_L))]$$

$$b_1 = \frac{V_1 - Z_0 I_1}{\sqrt{8}\, Z_0} = \frac{jV_0^+}{\sqrt{8Z_0}}[(1 - \Gamma_L) - \frac{Z_0}{Z_1}(1 + \Gamma_L))]$$

$$b_2 = \frac{V_2 - Z_0 I_2}{\sqrt{8}\, Z_0} = \frac{2V_2}{\sqrt{8Z_0}} = \frac{2V_0^+}{\sqrt{8Z_0}}(1 + \Gamma_L)$$

Por otra parte, las expresiones entre paréntesis son

$$1 + \Gamma_L = \frac{2Z_0}{Z_0 + Z_1}$$

$$1 - \Gamma_L = \frac{2Z_1}{Z_0 + Z_1}$$

En definitiva, obtenemos las ondas de potencia en función únicamente de las impedancias y de la amplitud V_0.

$$a_1 = \frac{2jV_0^+}{\sqrt{8Z_0}(Z_0+Z_1)}(Z_1 + \frac{Z_0^2}{Z_1})$$

$$b_1 = \frac{2jV_0^+}{\sqrt{8Z_0}(Z_0+Z_1)}(Z_1 - \frac{Z_0^2}{Z_1})$$

$$b_2 = \frac{2V_0^+}{\sqrt{8Z_0}(Z_0+Z_1)}2Z_0$$

Sustituyendo en la definición de parámetros [S] obtenemos:

$$S_{11} = \frac{b_1}{a_1}|_{a_2=0} = \frac{Z_1 - Z_0^2/Z_1}{Z_1 + Z_0^2/Z_1} = \frac{Z_1^2 - Z_0^2}{Z_1^2 + Z_0^2}$$

$$S_{21} = \frac{b_2}{a_1}|_{a_2=0} = \frac{2Z_0}{j(Z_1 + Z_0^2/Z_1)} = \frac{-2jZ_1^2Z_0^2}{Z_1^2 + Z_0^2}$$

La matriz de parámetros [S] completa es, por tanto:

$$[S] = \frac{1}{Z_1^2+Z_0^2}\begin{bmatrix} Z_1^2 - Z_0^2 & 2jZ_0Z_1 \\ -2jZ_0Z_1 & Z_1^2 - Z_0^2 \end{bmatrix}$$

Puede demostrarse fácilmente que la red es sin pérdidas. Por último, se va a particularizar el resultado para el caso de $Z_1 = 2\sqrt{Z_0}$

$$[S] = \begin{bmatrix} \frac{1}{3} & -j\frac{2}{\sqrt{2}} \\ -j\frac{2}{\sqrt{2}} & \frac{1}{3} \end{bmatrix}$$

Podemos acudir al Pozar para obtener una expresión general del parámetro S_{11} en función de la longitud eléctrica, y cuya justificación, se da en la misma sección.

$$|\Gamma| = \frac{|Z_L - Z_0|}{2\sqrt{Z_0 Z_L}} cos\,|\theta|$$

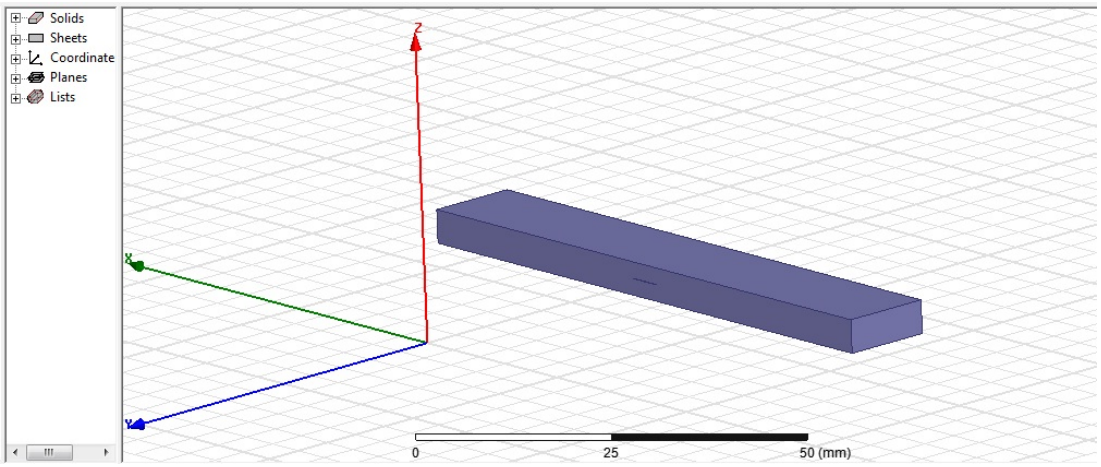


Figura 4.4: Linea de transmisión con HFSS.

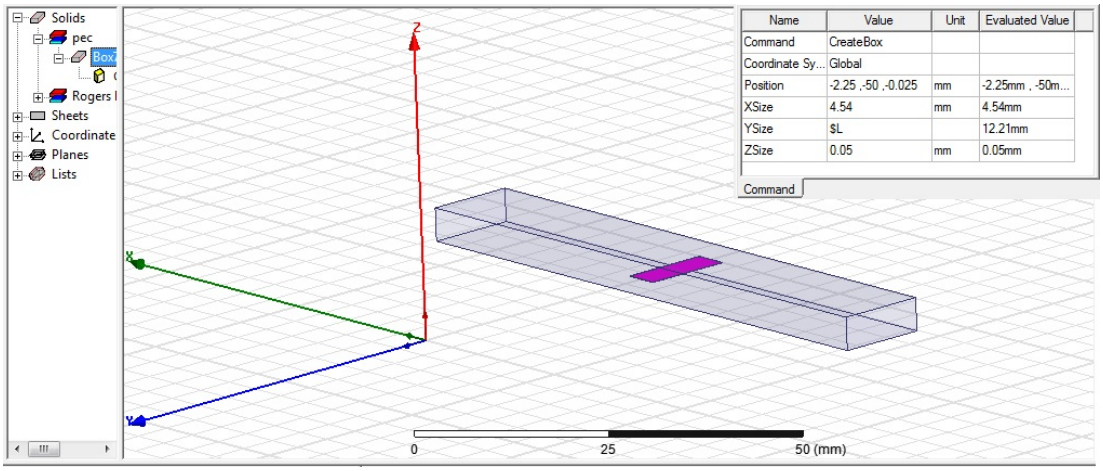


Figura 4.5: Dimensiones conductor central.

Esta función se puede codificar en matlab de forma inmediata y se ha validado con MWO [19].

Lo siguiente es diseñar la línea de transmisión a 12GHz con impedancia característica $Z_1 = 2\sqrt{2}$, las dimensiones de la estructura se pueden calcular mediante las fórmulas analíticas o cualquier calculador existente en la red.

En las Figuras 4.5, 4.6 y Figura 4.7 se muestra la estructura diseñada junto a sus dimensiones, donde L (longitud eléctrica de la línea de transmisión), es el parámetro a optimizar.

Tras el diseño de la estructura electromagnética, se procede a la optimización de la misma. Para ello inicializamos el espacio de búsqueda considerando un solo

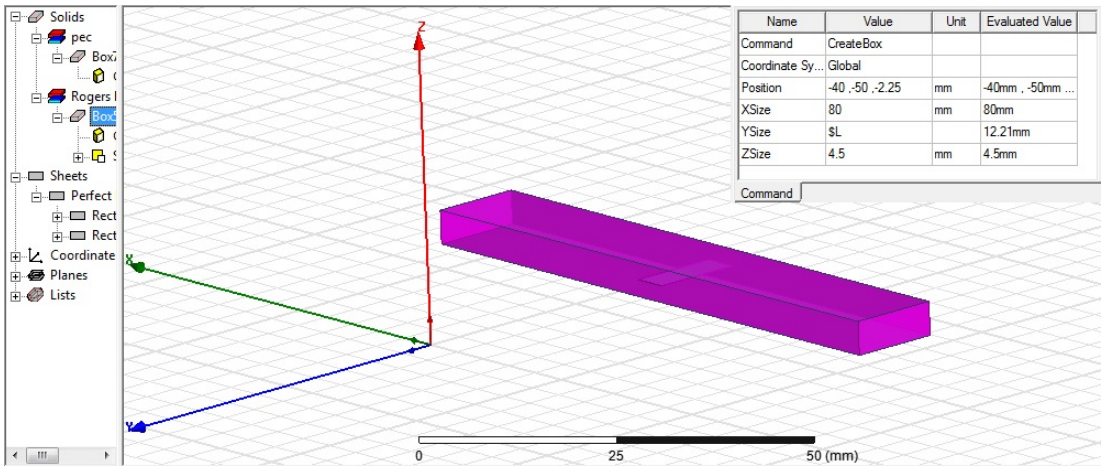
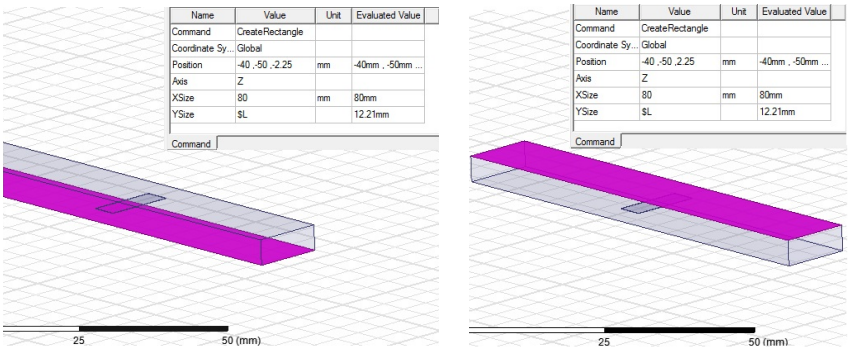


Figura 4.6: Dimensiones del tramo de linea.



(a) Dimensiónes $z = -2{,}25mm$ (b) Dimensiónes $z = 2{,}25mm$

Figura 4.7: Dimensiones planos en z

mínimo. El rango de valores propuesto es:

$$l1 \pm \lambda/5$$

Donde por ser periódica:

$$l1 = \lambda/4 + \lambda/2$$

Tras lo anterior se van a mostrar los resultados obtenidos para cada iteración, observando cómo el algoritmo de búsqueda encuentra la solución óptima en un número de pasos determinado.

Debemos resaltar que los resultados se muestran como magnitud respecto longitud, puesto que, es este último, el parámetro que queremos optimizar. Además, obtenemos en cada análisis un único punto en frecuencia, siendo innecesario, en este caso, un sweep, puesto que, para la frecuencia de operación optimizamos la estructura a través de la longitud. El sweep también esta implementado para otros posibles escenarios.

Es importante notar que se ha etiquetado el número de individuos obtenidos, de forma que podemos ir viendo progresivamente cómo el algoritmo se queda con los individuos más aptos.

La precisión escogida para este análisis no ha sido muy elevada n=3:

En la Figura 4.8 se muestran las dos primeras etapas del algoritmo:

- En '*Inicialización*' se muestra la población aleatoria inicial, conjunto de soluciones candidatas.
- '*Primera selección*' ya ha obtenido mediante una simulación paramétrica distribuida los resultados de los análisis de los invidivuos, y tras ello, se han realizado las operaciones de evaluación, reproducción, cruce y mutación. El algoritmo comienza a considerar los individuos más aptos. Recuérdese que el mecanismo de selección es por torneo en versión probabilística.

En la Figura 4.9 se muestran las dos siguientes etapas, con los siguientes resultados:

- '*Segunda selección*' es la primera iteración del algoritmo, en ella se muestra que el algoritmo ha desechado los individuos menos aptos (los más alejados de la solución óptima). Como el individuo más numeroso no es el óptimo se realiza una nueva iteración.
- '*Tercera selección*' es la segunda iteración del algoritmo, en ella se muestra cómo el algoritmo encuentra la solución óptima, seleccionando mayoritariamente el individuo más apto.

El algoritmo puede ser parado en cualquier momento por el usuario, seleccionando como solución de optimización el individuo mayoritario. El individuo seleccionado es el que tiene menor distancia con respecto a 12,64mm, que es la solución teórica óptima justificada en la subsección de validación de este capítulo.

Todos los análisis se realizan de forma ditribuida en el cluster del departamento, obteniendo así, altas prestaciones de cómputo. Esto reduce el tiempo de ejecución drásticamente.

El individuo mayoritario da como solución 13,5mm, que es la solución más cercana a 12,64mm. Es de notar que el algoritmo no ha encontrado el valor teórico óptimo porque no estaba entre el conjunto de soluciones candidatas. Esto se solucionaría aumentando la resolución del parámetro o disminuyendo la frecuencia de paso en cada nueva iteración del algoritmo, forzando a este a dar con el valor exacto teórico.

4.3. Validación

El transformador diseñado tiene una frecuencia de operación de 12GHz. A partir de este dato se puede calcular la longitud eléctrica a $\lambda/4$:

$$\lambda = \frac{c_0}{f\sqrt{\epsilon}}$$

Donde f es la frecuencia de operación (12GHZ) y ϵ es la permitividad relativa del dieléctrico escogido, $\epsilon = 2{,}2$.

De lo anterior se deduce $\lambda/4 = 4{,}21mm$, se ha optado por coger $\lambda/4 + \lambda/2 = 12{,}64mm$ puesto que es sabido que arroja los mismos resultados.

El rango de valores teórico sobre el que se seleccionan aleatoriamente las soluciones candidatas que conforman el espacio de búsqueda es:

$$l1 = \lambda/4 + \lambda/2$$

$$limitesup = l1 + \lambda/5 = 16{,}01mm$$

$$limiteinf = l1 - \lambda/5 = 9{,}27mm$$

Para el rango de valores dado se obtiene la mínima amplitud del parámetro [S] en el punto de operación del transformador, y al acercarnos progresivamente a los límites, experimentamos un aumento.

En este caso se va a estudiar el parámetro S_{11}. Recuérdese que su valor se describe con la siguiente fórmula:

$$|\Gamma| = \frac{|Z_L - Z_0|}{2\sqrt{Z_0 Z_L}} cos\,|\theta|$$

Donde Z_L y Z_0 se definen como muestra la Figura 4.3 y donde $\theta = \beta\, l$.

Se implementa pues, dicha fórmula en Matlab para obtener la solución teórica del parámetro S_{11}:

Fichero 4.5: lambda4.m

```
f0 = 12e9;
c= 3e8;
nbits=8;
nbits=8;
landa0 = c/(f0*sqrt(2.2));
f1=linspace(1e9,40e9,2^nbits);
landa1 = c ./ (f1*sqrt(2.2));
beta = 2*pi./landa1;
l = landa0/4;

sopt = ( abs(zL-z0)/( 2*sqrt(z0*zL) ) )* abs(cos(beta*l));
sopt = 20*log10(sopt);
plot(f1, sopt)
```

La frecuencia *f1* del fichero 4.5 se ha fijado desde *1GHz* hasta *30GHz*. Los resultados de ejecución indican que $S_{11} = -63dB$ a la frecuencia de trabajo y $S_{11} = -15dB$ a 2 veces la frecuencia de trabajo. Ver Figura 4.10.

Esto puede deducirse directamente de la fórmula teórica vista, puesto que:

$$\theta = \beta\, l$$

De donde:

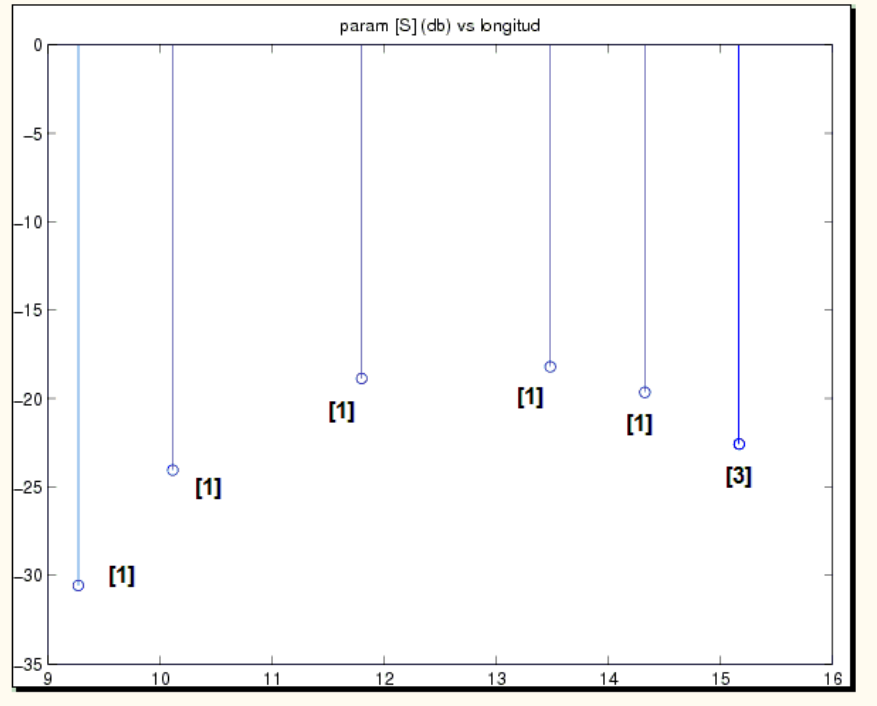
$$\beta = \frac{2\pi}{\lambda}$$

$$l = \frac{\lambda_0}{4}$$

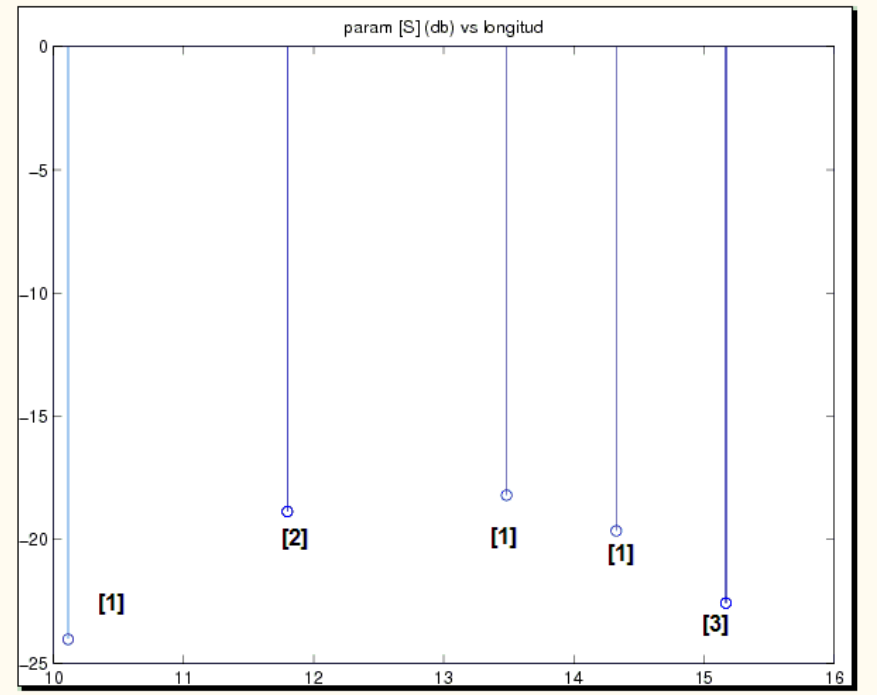
De esta forma podemos deducir directamente:

$$\theta = \frac{\pi f_1}{2 f_0}$$

Si sustituimos f_1 por f_0 y $2f_0$ se obtienen los mínimos de la función anterior. Por lo que se puede deducir que la figura obtenida con Matlab es válida. Aún así, se han validado con MWO [19] los resultados obtenidos.

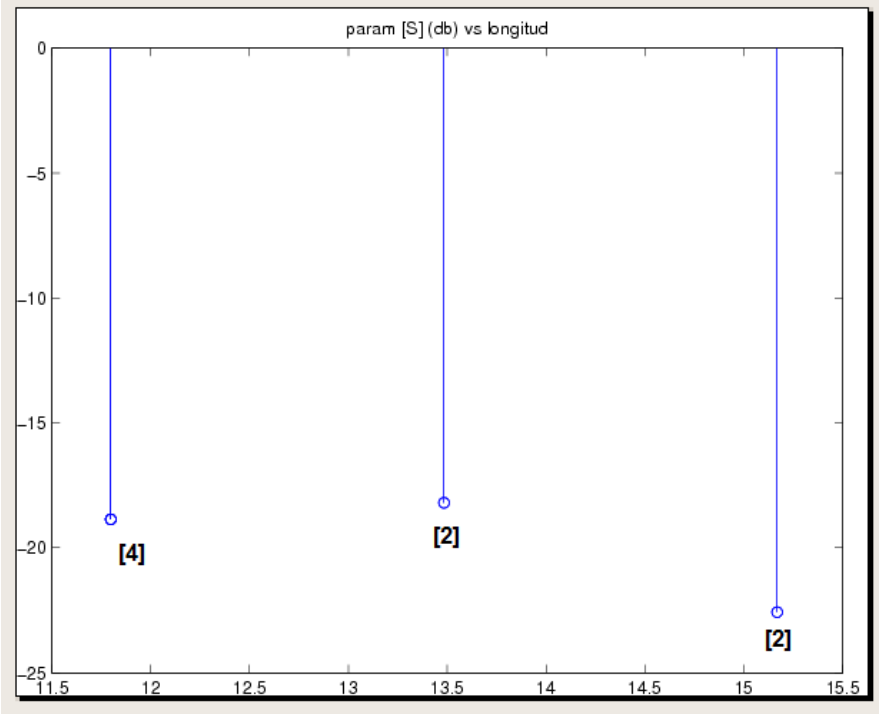


(a) Inicialización

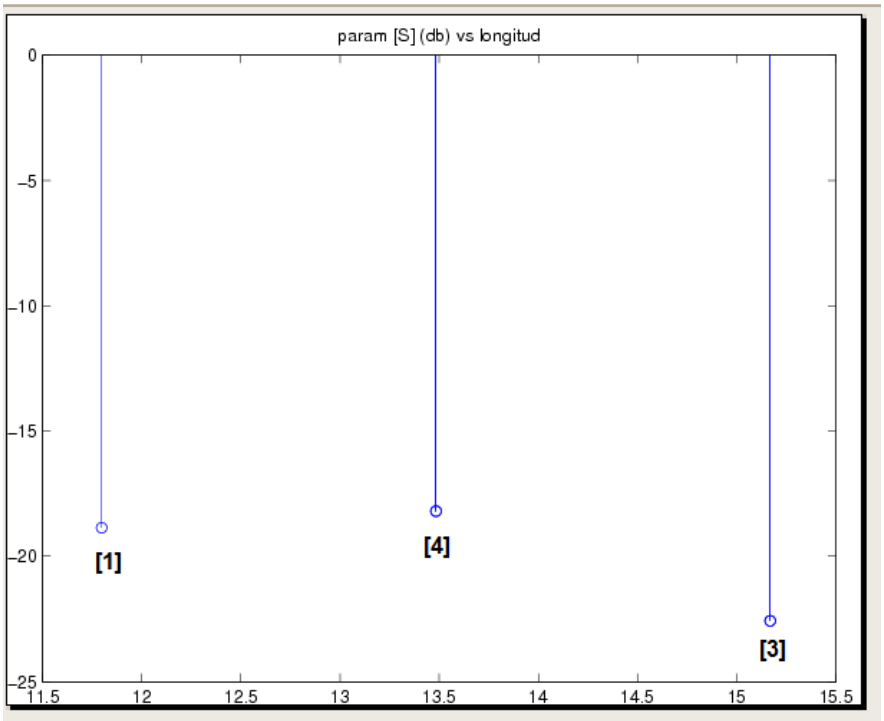


(b) Primera Selección

Figura 4.8: Resultados de la primera iteración



(a) Segunda Selección



(b) Tercera Selección

Figura 4.9: Resultados de la segunda y tercera iteración

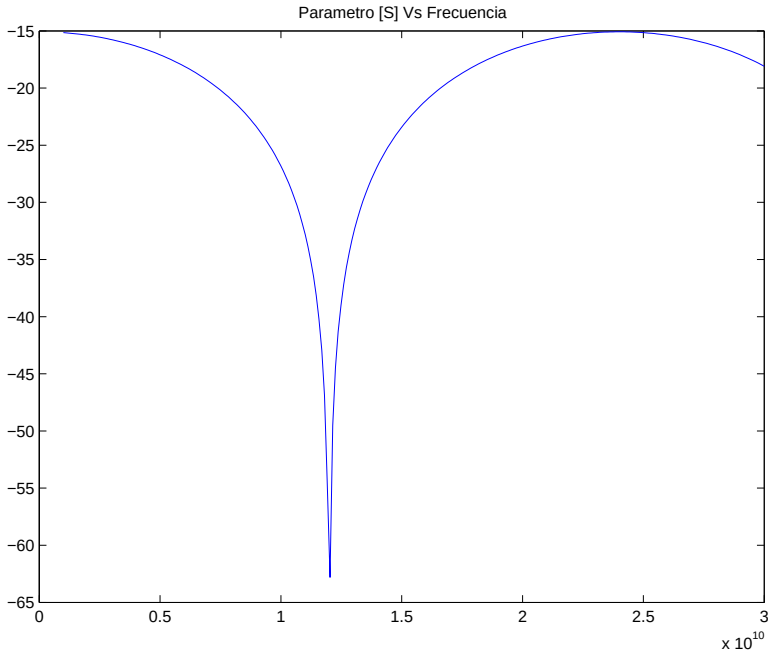


Figura 4.10: Parámetros [S] con Matlab.

Capítulo 5

Sistema completo: Optimización de Filtro Dieléctrico

En el presente capítulo presentamos la optimización del Filtro Dieléctrico descrito en el capítulo 3. Para ello, se debe determinar el escenario de optimización cuya inicialización debe ser descrita por el usuario.

Tras la inicialización se configura el sistema completo, el cual se compone de todas las herramientas diseñadas en los capítulos 3 y 4 tal y como se muestra en la Figura 5.1.

La Figura 5.1 está compuesta por una caja inicial (Inicialización) y final (Evaluación, reproducción, cruce y mutación), que implementan el algoritmo genético a través de las funciones ahí mismo referenciadas. La gran capacidad de paralelización de este algoritmo permite la simulación parámetrica distribuida, compuesta por las cajas restantes.

Para implementar este flujo de ejecución, se requiere el uso de diversos programas software que son llamados a través de múltiples scripts. Se necesita por tanto, un script que ejecute cada una de las partes del sistema de la Figura 5.1 y que denominaremos *maestro.sh*.

Fichero 5.1: maestro.sh

```
#!/bin/ksh

./clear.sh
qdel -u cmlentis

/opt/matlab/bin/matlab -nosplash -nodisplay -nojvm -r inicializacion

i=1
```

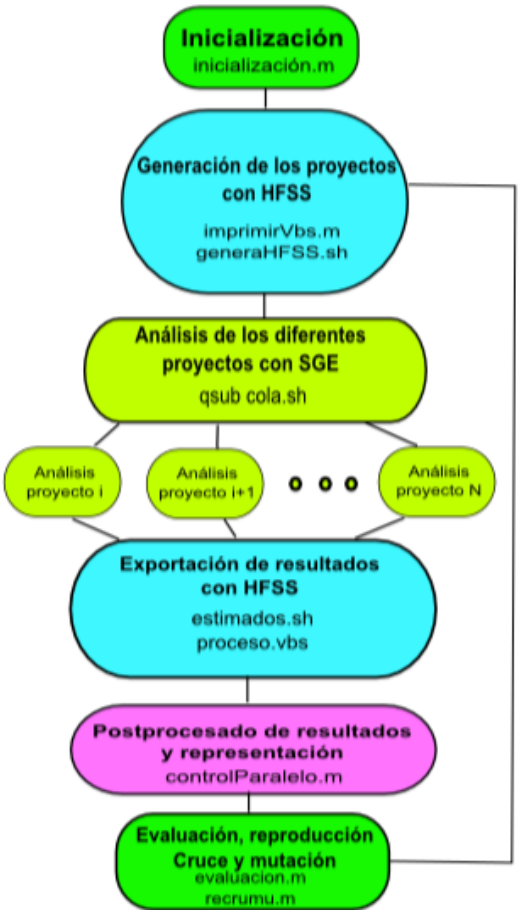


Figura 5.1: Diagrama de flujo del sistema y funciónes asociadas

```
   nVeces=3

   while [ $i -le $nVeces ]; do
12
   ./clear2.sh
   /opt/matlab/bin/matlab -nosplash -nodisplay -nojvm -r generoproyecto
   ./generaHFSS 16

17 qsub cola.sh

   MECANISMO DE ESPERA DE QSUB

   MECANISMO DE MONITORIZACIN FICHEROS .log
22
   ./estimados.sh

   /opt/matlab/bin/matlab -nosplash -nodisplay -nojvm -r controlParalelo
   /opt/matlab/bin/matlab -nosplash -nodisplay -nojvm -r evalua
27 /opt/matlab/bin/matlab -nosplash -nodisplay -nojvm -r recrumu

   i=$(($i+1))

   rm -r *.log
32
```

```
done
```

La línea 3 del fichero *maestro.sh* sirve para limpiar el directorio de trabajo, la línea 4 borra de la cola las tareas pendientes que pudiese haber antes de iniciar nuestro sistema.

A continuación, se itera el algoritmo para refinar la solución, pudiéndose modificar el número de iteraciones que se realizan. Antes de comenzar una nueva iteración borramos con el script *clear2.sh* los ficheros *.vbs* generados en iteraciones pasadas.

Con las líneas 14 y 15 generamos los N subproyectos a través de los N ficheros *.vbs* escritos por la función *imprimirVbs.m*, y lanzamos con *qsub* (linea 17) la tarea al sistema gestor de colas. Tras pasar por los mecanismos de monitorización descritos en el capítulo 3, se obtienen los reports con el script *estimados.sh* y se pasa al postprocesado de la información, línea 25.

Tras obtener los resultados de la simulación paramétrica, el sistema recurre al algoritmo genético (líneas 26 y 27), el cual se compone de evaluación, reproducción, cruce y mutación, obteniendo finalmente, los individuos supervivientes que componen la siguiente generación de la población.

Por último, se borran los ficheros existentes *.log* para evitar conflictos.

Se debe destacar que ha sido necesario incluir en el sistema una nueva función. Esta función imprime a disco el resultado de la evaluación del algoritmo genético. De esta forma, el usuario tiene conocimiento exacto de cuáles son los individuos que componen la población actual y cuál es el valor de sus parámetros.

El análisis de estos ficheros de salida en conjunto con las imágenes obtenidas, dan al usuario una visión completa de la optimización de la estructura, pudiéndose determinar, por ejemplo, cuántos individuos pertenecen a la solución óptima tras la finalización del algoritmo (como se presentó en el capítulo 4), o cuál es el valor óptimo de los parámetros del individuo seleccionado como solución óptima.

A continuación, se va a observar cómo nuestro sistema alcanza la solución óptima progresivamente, convergiendo a dicha solución.

5.1. Análisis de los resultados

El usuario debe definir en la *inicialización* cierta información:

- El número de bits que codifican cada parámetro que se desea optimizar.
- El nombre que adquiere dicho parámetro en la estructura HFSS dada.
- Valor inicial que adquiere el parámetro y unidades de medida para dicho valor.
- Información relativa al rango de valores que puede tomar un parámetro en la optimización.
- Identificador del proyecto a optimizar.
- Número de individuos que componen la población inicial.

En nuestro caso, se ha definido una optimización del filtro dieléctrico basada en una población de 16 individuos. Los parámetros a optimizar son las dimensiones físicas de las ventanas de acoplo, definidas como *w2,w3,h2 y h3*, longitudes en los ejes Y y Z respectivamente (ver Figura 3.3).

Previa a la optimización de la estructura, se ha fijado la altura de los resonadores para que la frecuencia de operación sea *11.6GHz*. Además, se ha determinado utilizar 4 bits para codificar los parámetros *w2,h2* y 3 bits para los parámetros *w3,h3*.

Una vez fijado el rango de valores, rango que se discretizará para obtener 2^N posibles soluciones de dicho parámetro, se determina el objetivo final. Este objetivo consiste, para el caso del filtro dieléctrico, en fijar su máscara.

La máscara deseada por el usuario para el parámetro S_{11} es de -20dB en la banda de paso y 0dB fuera de la banda de paso. El optimizador diseñado seleccionará la solución que más se adecua a dicha necesidad definida por el usuario, usando, para ello, una población de 16 individuos, pero que bien podría haber sido superior (recuérdese que podemos aumentar el grado de precisión de la solución óptima sin más que aumentar el conjunto de posibles soluciones, es decir, aumentar el conjunto poblacional).

Con las anteriores consideraciones se inicia el optimizador. Como se estudió en capítulos anteriores, el sistema debe converger a la solución óptima, solución que

más se acerca a la deseada (máscara del filtro definida por el usuario). Analizando la figura 5.2 se ve este comportamiento.

En la figura 5.2 se presentan los diez primeros ciclos de simulación, resultados que el algoritmo genético va obteniendo iterativamente. Cada iteración ha sido etiquetada con un valor, fíjese que en figuras consecutivas no se aprecia de forma cualitiva que el sistema esté realmente despreciando las soluciones menos óptimas, sin embargo, si el lector observa en primer lugar la subfigura 1, y a continuación, la subfigura 10, podrá observar cómo el sistema se está comportando correctamente, puesto que, pasadas 10 iteraciones, el sistema ha despreciado muchas de las soluciones candidatas en la población inicial.

En cada ciclo de simulación se realizan 16 análisis, distribuidos gracias al gestor de colas en el cluster del Grupo de Radiofrecuencia del departamento de Teoría de la Señal y de las Comunicaciones. Es lo anterior donde se aporta más valor, puesto que, reducimos considerablemente, el tiempo de ejecución que se necesita para una optimización de estas características.

Una vez que el sistema se acerca a un conjunto más reducido de posibles soluciones, resulta conveniente visualizar cuantitativamente cómo el sistema converge a la solución óptima. Esto sucede por la propagación en la población del individuo cuyos parámetros son óptimos, es decir, las soluciones no óptimas irán desapareciendo progresivamente de la población, mientras que, la solución óptima se propagará de forma que la solución final tendrá, idealmente, solo el individuo óptimo o superviviente.

En la figuras 5.3, 5.4, 5.5 se observa dicho comportamiento:

1. En la figura 5.3 se observa:

   - 7 individuos con el identificador número 3.
   - 2 individuos con el identificador número 2.
   - 5 individuos con el identificador número 7.
   - 2 individuos con el identificador número 8.

   Es decir, tenemos una alta presencia del individuo 7 y 3 y baja presencia del individuo 2 y 8, individuos que serán despreciados en las siguientes iteraciones.

2. En la figura 5.4 se observa:

   - 7 individuos con el identificador número 3.

- 5 individuos con el identificador número 7.
- 4 individuos con el identificador número 8.

Como se puede deducir de lo anterior, el individuo con identificador número 2 ha sido desechado por el algoritmo ya que es una solución subóptima.

3. En la figura 5.5 se observa:

   - 14 individuos con el identificador número 3.
   - 2 individuos con el identificador número 8.

   De lo anterior se deduce que el individuo con identificador 7 es, también, una solución subóptima que el algoritmo ha desechado, y que el individuo con identificador número 3 es la solución dada por el algoritmo en 34 iteraciones.

Una vez que se ha encontrado la solución superviviente, solo queda observar los valores de los parámetros, valores que se adjuntan en la figura 5.6, que representa la solución final.

Para finalizar, conviene caracterizar el filtro completamente representando el parámetro S_{21} que, conjuntamente con el parámetro S_{11}, se observa en la figura 5.7.

La solución presentada es, sin embargo, una solución subóptima con respecto a la solución teórica deseada [20]. A pesar de que todos los individuos recorren el espacio de búsqueda en paralelo, el algoritmo no ha sido capaz de encontrar la solución óptima en 34 iteraciones. Esto se debe a:

- El número de individuos que componen la población.

  En este caso se ha inicializado una población de 16 individuos con respecto a los 2^N individuos que componen el conjunto de soluciones candidatas. Aumentando el conjunto poblacional se aumentará la potencia de búsqueda del algoritmo, es decir, un conjunto poblacional pequeño requerirá mas iteraciones (reproducción, cruce, mutación) para explorar el espacio de búsqueda, que una población formada por un número mayor de individuos.

- Selección de los parámetros de optimización.

  La correcta selección de los parámetros es fundamental para encontrar los resultados teóricos deseados. Para el análisis descrito, se localiza un mínimo absoluto, este mínimo es, sin embargo, un mínimo relativo si aumentamos el número de parámetros a optimizar. Por esta razón es importante determinar los parámetros suficientes y necesarios para encontrar la solución óptima y no llegar a una solución subóptima como en el caso que se presenta. Es importante también destacar que gracias al diseño definido, el usuario no tiene limitaciones con respecto al número de parámetros a optimizar, siempre podría realizar un ataque de fuerza bruta en el peor de los casos.

- Codificación de los parámetros.

  Aumentar el número de bits que codifican los parámetros es también una forma de encontrar la solución óptima. No solo su correcta selección, sino también su correcta codificación permiten al algoritmo encontrar el mínimo absoluto. El filtro aquí descrito requiere de una gran precisión, por lo que si se desea explorar un rango valores considerable, el número de soluciones candidatas también crece considerablente. El problema de lo anterior, como se puede intuir, es la gran carga computacional que se deduce de ello.

La solución presentada podría, por tanto, mejorarse aumentando el número de parámetros a optimizar, codificando estos de una forma más precisa y localizada en torno al valor óptimo. El no conocimiento preciso de los valores óptimos incrementa, considerablemente, el espacio de búsqueda y la carga computacional del algoritmo para encontrar la solución, por esta razón, no se ha inicializado el algoritmo para realizar un ataque de fuerza bruta.

Sin embargo, se debe resaltar que la solución dada no está lejos de la solución óptima (figura 5.8). Esta solución óptima y la metodología para llegar a ella, se detallan profundamente en el artículo [20].

El único problema que presenta la solución dada es la atenuación introducida en la transmisión de energía para frecuencias cercanas a la frecuencia de corte inferior. El resultado podría refinarse en función de las necesidades del usuario. Como se puede ver, la metodología descrita da total libertad al usuario para la realización de la optimización.

Conviene realizar una consideración sobre lo aquí descrito. En la optimización realizada, se parte de una población de N individuos con N constante a lo largo de la optimización. Todos los individuos son susceptibles de ser transformados en los procesos de reproducción, cruce y mutación. Esto ha sido conscientemente realizado para obtener el estudio de convergencia de toda la población en paralelo, observando como todos y cada uno de los individuos convergen a la solución óptima.

Sin embargo, es posible aumentar la eficiencia del algoritmo eliminando progresivamente individuos del conjunto de soluciones candidatas. De esta forma, un individuo no podrá transformarse en un individuo cuyo fitness sea menor bajo ninguna probabilidad. Esto aumenta la velocidad de convergencia del optimizador.

Aumentar excesivamente la velocidad de convergencia del optimizador puede generar un problema: no se explora eficientemente el espacio de búsqueda y los

superindividuos no pertenecen a una búsqueda global, sino a una búsqueda local. Se ha encontrado un mínimo local de la función de coste y no el mínimo absoluto.

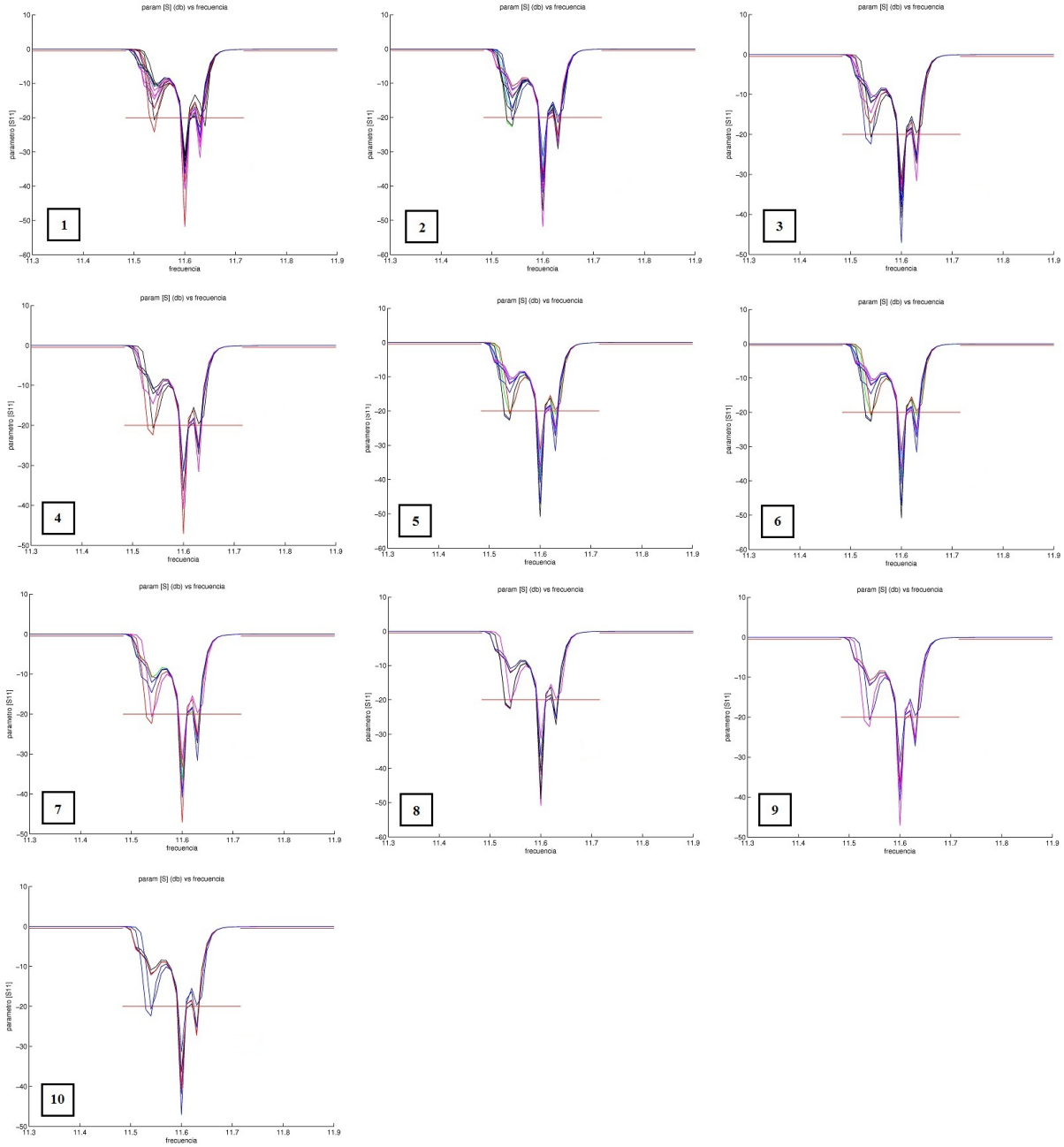


Figura 5.2: Primeros diez ciclos de simulación

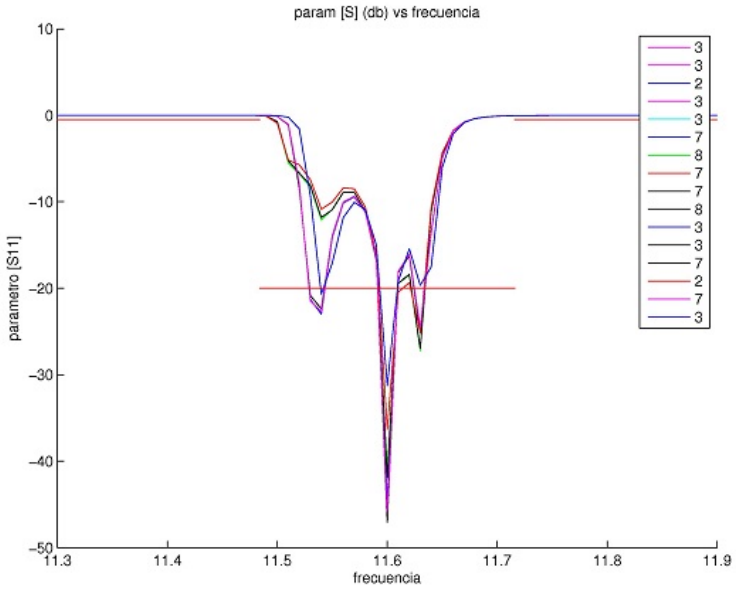


Figura 5.3: Últimos ciclos de simulación (I)

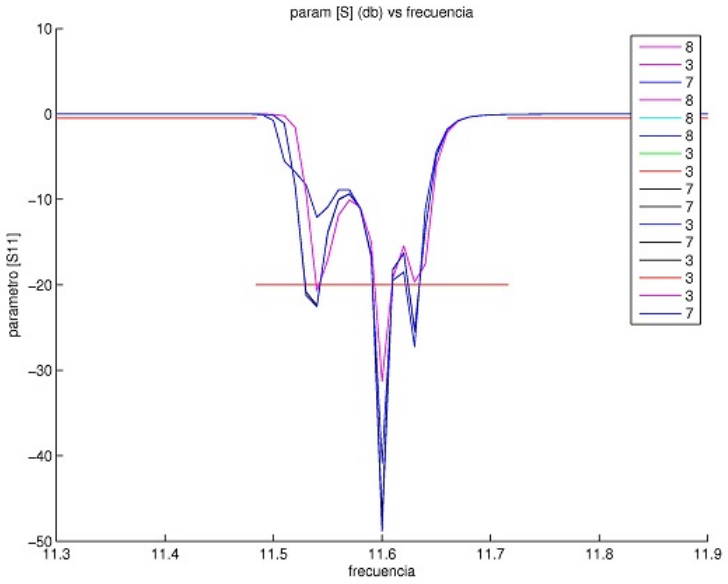


Figura 5.4: Últimos ciclos de simulación (II)

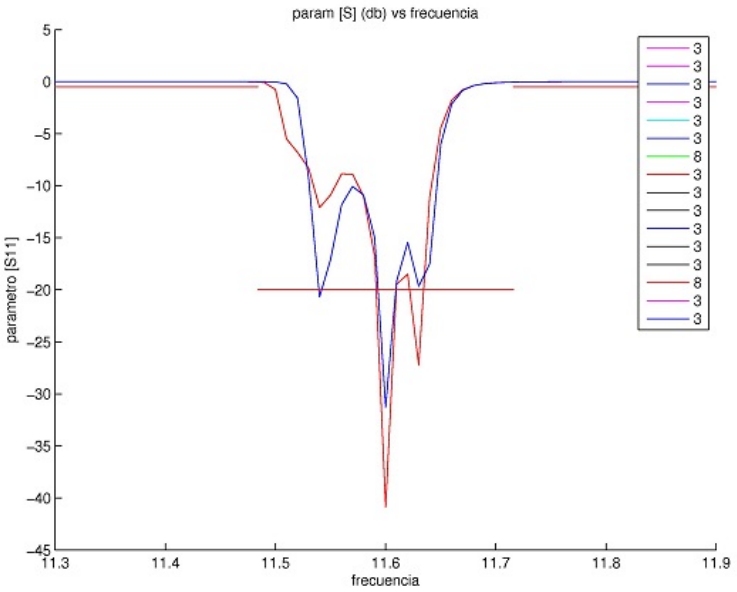


Figura 5.5: Últimos ciclos de simulación (III)

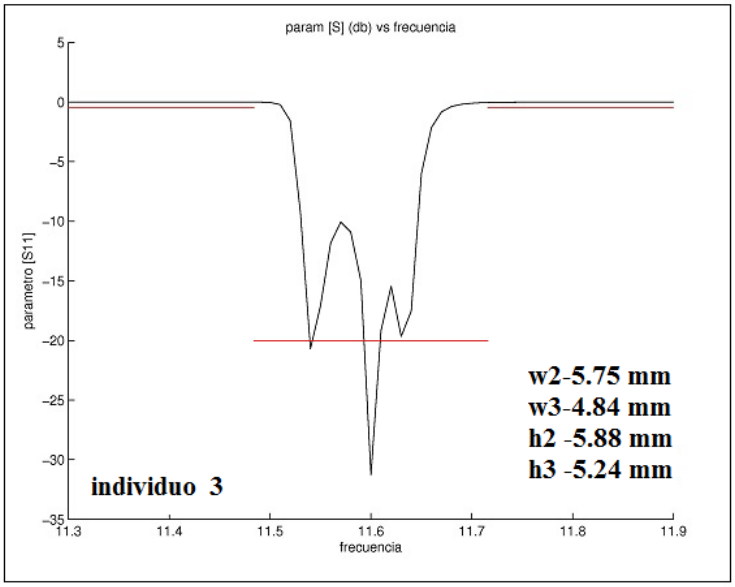


Figura 5.6: Solución final: Valor óptimo de los parámetros

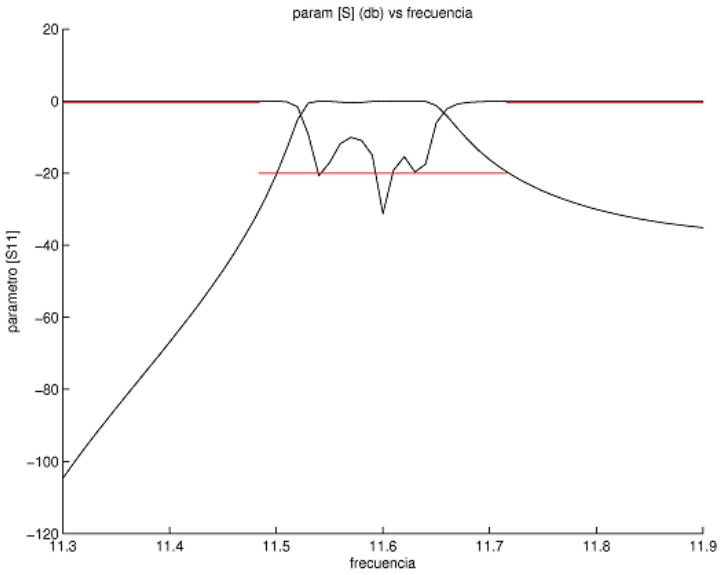
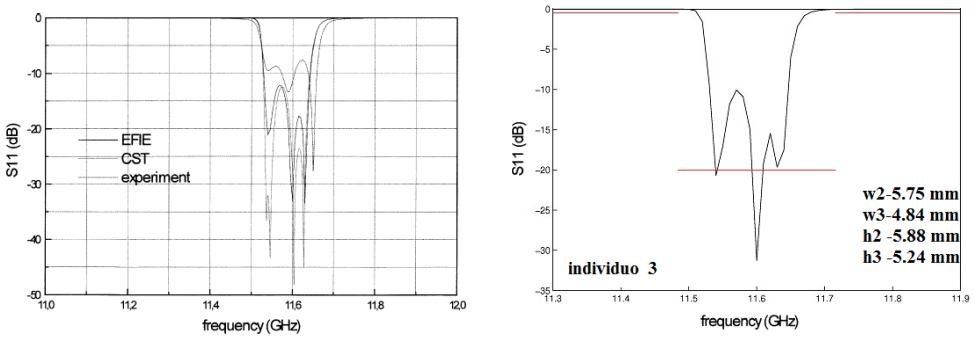


Figura 5.7: Solución final. Parámetros S_{11}, S_{21}



(a) Respuesta óptima (b) Herramienta de optimización

Figura 5.8: Respuesta óptima [20] Vs Herramienta de optimización

Capítulo 6

Desarrollo del PFC

El Proyecto Fin de Carrera comienza con una propuesta inicial donde se presenta al proyectante la motivación y finalidad del mismo. Tras lo anterior, se marcan los objetivos.

Una vez presentado, se indican las herramientas software que son necesarias para su realización. La primera fase comprende el estudio de las diversas herramientas, en particular HFSS y SGE, herramientas sobre las que se partía del desconocimiento del proyectante. Este estudio implica la lectura de manuales de fabricante y la interacción con dichas herramientas mediante ejemplos sencillos y de carácter didáctico.

La adquisición de conocimientos básicos sobre la funcionalidad de HFSS y SGE se complementa con el estudio de VBScripting, lenguaje de programación que utiliza HFSS para generar cualquier diseño, análisis y obtención de resultados, y que resulta necesario para la resolución de este proyecto.

La necesidad de trabajar en remoto requiere la configuración del entorno para que el proyectante interactue con el sistema en un modo no presencial, esto es un requisito del sistema. Cualquier usuario conectado a Internet debe poder hacer uso de la herramienta diseñada. Esta configuración se realiza en paralelo con el estudio de las herramientras de trabajo.

La configuración del entorno plantea una serie de problemas, como la configuración de la VPN a la red del departamento o la incompatibilidad del software, que ocasiona que la interfaz gráfica de HFSS no sea visible al proyectante. Este problema imposibilita el progreso del PFC y se soluciona con la herramienta NX Client, que sustituye a Xming y Putty.

Gracias a la configuración anterior, el proyectante puede proceder a desarrollar la metodología de simulación paramétrica distribuida propuesta, implementando los scripts que gobiernan el comportamiento de HFSS y SGE para la generación de proyectos, análisis y obtención de resultados.

Pese a los problemas que se derivan en el planteamiento inicial, se replantea la metodología, dividiéndose esta, en las tres fases aquí documentadas. Esto obliga al proyectante a rehacer todo el desarrollo. Las tres fases que componen la simulación paramétrica distribuida son realizadas en paralelo.

Una vez implementada la metodología se procede a la validación de la misma, esto requiere un proceso de depuración de errores que finaliza satisfactoriamente. Para validar la metodología, se utiliza un filtro dieléctrico de 4 cavidades, estructura diseñada paramétricamente y que es estudiada para su comprensión, poniéndose especial interés en los parámetros que gobiernan su comportamiento.

Una vez obtenido el análisis paramétrico del filtro, se redacta, con ayuda del director, el artículo presentado al XXVII Simposium Nacional de la Unión Científica Internacional de Radio. Esto supone una primera redacción rigurosa sobre el trabajo realizado, trabajo que por otra parte, es documentado progresivamente en el wiki del Grupo de Radiofrecuencia.

En este punto se extiende el objetivo con el desarrollo de una herramienta de optimización, lo que nos permite ilustrar la metodología propuesta. Esta herramienta utiliza un algoritmo genético como algoritmo de búsqueda, cuya justificación requiere el estudio de otros algoritmos existentes. El desarrollo del algoritmo genético también requiere, en cada fase, la justificación de la estrategia empleada. Tras lo cual, se realiza una primera implementación para encontrar un mínimo en una función matemática sencilla.

Posteriormente, se integra el optimizador en el sistema completo, lo que permite utilizar la simulación paramétrica distribuida en el proceso de optimización. Esto es, sin dudas, una fase importante del proyecto. Para validar la herramienta de optimización se diseña un transformador $\lambda/4$, estructura ampliamiente conocida y que además, se resuelve teóricamente.

Tras lo anterior se comienza la redacción del presente Proyecto Fin de Carrera, tarea que se realiza, en un primer momento, en paralelo con la optimización del filtro usado para validar la metodología de simulación paramétrica distribuida. En este punto, se sugiere que el sistema es válido pero requiere un alto conocimiento del usuario de cada una de las partes diseñadas, por ello, se vuelve a modificar el sistema, de forma que, el usuario solo deba introducir unos datos

de entrada y pueda obtener los resultados deseados, esto se consigue mediante la automatización de tareas.

Finalmente, y con las recomendaciones del director/tutor, se realizan una serie de revisiones que concluyen con el documento aquí presentado.

El diagrama de Gantt de la Figura 6.1 representa el desarrollo descrito.

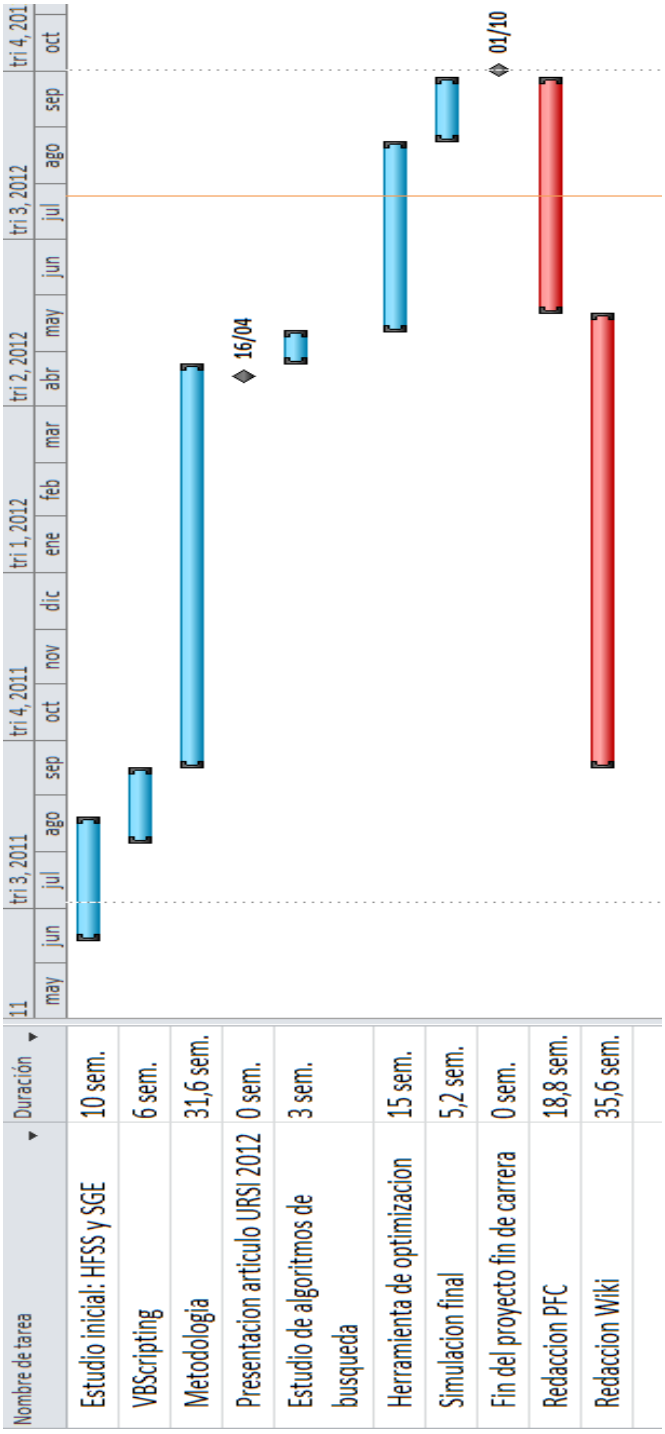


Figura 6.1: Planificación: Diagrama de Gantt.

Capítulo 7

Conclusiones y líneas futuras

7.1. Conclusiones

En primer lugar, se debe destacar la flexibilidad de la metodología de simulación paramétrica distribuida propuesta, ya que el diseñador puede determinar el número de parámetros de la estructura a modificar y seleccionar cada uno de ellos. Además, el diseñador puede decidir el número de modificaciones sobre cada uno de los parámetros elegidos convirtiendo su análisis paramétrico en N análisis.

El análisis de los resultados obtenidos por el sistema que hace uso de esta metodología , permite concluir que el tiempo de ejecución se ha reducido notablemente, gracias al uso eficiente de los recursos hardware que se disponen. La resolución secuencial de 125 simulaciones de un filtro dieléctrico de 4 cavidades en guía onda, dura, aproximadamente, 49 horas en un ordenador de un solo núcleo, doce horas y media en un Intel Core 2 Quad, y 32 minutos en el cluster de cómputo científico del Grupo de Radiofrecuencia, cluster que dispone de veinte nodos con ocho núcleos cada uno.

La herramienta de optimización desarrollada hace uso de esta simulación paramétrica distribuida en el proceso de optimización y adquiere sus propiedades. De esta forma, se minimiza el tiempo de ejecución para encontrar la solución óptima. Para validar el optimizador se ha comprobado que el algoritmo es capaz de encontrar dicha solución.

La herramienta permite, además, que el usuario pueda definir cuántos y qué parámetros desea optimizar, así como orientar la búsqueda de sus valores óptimos, determinando el número de bits que codifica cada uno de ellos o el espacio de búsqueda

donde se localiza la solución óptima. Esta codificación permite fijar la precisión de dicha búsqueda.

Para mejorar la experiencia del usuario, se ha desarrollado el optimizador evitando, en la medida de lo posible, su interacción con la herramienta. Gracias a la automatización de tareas solo se requiere la inicialización del sistema y el objetivo de la optimización.

En este documento se ha optimizado un filtro dieléctrico de 4 cavidades en guía onda, pero la herramienta se ha validado para otros dispositivos de alta frecuencia, como por ejemplo: antenas, resonadores, etcétera.

7.2. Líneas futuras

Como línea futura se plantea modificar el algoritmo genético, para que en cada iteración, se refine el paso entre valor y valor del espacio de búsqueda. El objetivo es obtener con seguridad la solución óptima y no una aproximación, situación que sucede cuando esta solución óptima no se encuentra entre las soluciones candidatas.

El sistema realiza una espera activa que finaliza cuando HFSS detecta la finalización correcta en su flujo de ejecución, lo que ocurre para cada subproyecto analizado. Otra posibilidad consiste en proporcionar una espera por eventos, consiguiendo así, una mayor eficiencia y menor coste computacional.

Para mejorar la experiencia del usuario se plantea el diseño de estructuras predefinidas con VBScript. Estas estructuras podrían modelarse con VBScript indicando algunos parámetros de las mismas. Un ejemplo de esto sería un array de antenas, donde el usuario podría definir, entre otras opciones, el número de elementos del array.

Otra línea de investigación futura podría proporcionar una interfaz gráfica a la herramienta, de forma que, el usuario pudiese seleccionar las acciones mediante el uso de botones y menús.

Capítulo 8

Presupuesto

En este capítulo se presentan justificados los costes globales de la realización de este Proyecto Fin de Carrera. Tales costes, imputables a gastos de personal y de material, se pueden deducir en el siguiente resumen:

UNIVERSIDAD CARLOS III DE MADRID
Escuela Politécnica Superior

PRESUPUESTO DE PROYECTO

1.- Autor:

Carlos Mariano Lentisco Sánchez

2.- Departamento:

Teoría de la Señal y las Comunicaciones

3.- Descripción del Proyecto:

- Titulo: **Herramienta de optimización paramétrica distribuida con HFSS**
- Duración (meses): **18**
- Tasa de costes Indirectos: **20%**

4.- Presupuesto total del Proyecto (valores en Euros):

Euros: **183645**

5.- Desglose presupuestario (costes directos)

PERSONAL

Apellidos y nombre	Categoría	Dedicación (hombres mes) [a]	Coste hombre mes	Coste (Euro)
Carlos Mariano Lentisco Sánchez	Ingeniero Junior	8	1.800,00	14.400,00
Ignacio Martínez Fernández	Administrador Senior	4	2.800,00	11.200,00
	Hombres mes 12		**Total**	25.600,00

[a] 1 Hombre mes = 131,25 horas. Máximo anual de dedicación de 12 hombres mes (1575 horas)
Máximo anual para PDI de la Universidad Carlos III de Madrid de 8,8 hombres mes (1.155 horas)

EQUIPOS

Descripción	Coste (Euro)	% Uso dedicado proyecto	Dedicación (meses)	Periodo de depreciación	Coste imputable [d]
Laptop	700,00	100	14	36	272
Cluster del grupo de radiofrecuencia	163.498,00	100	14	36	63.583
				Total	63.854,78

[d] Fórmula de cálculo de la Amortización:

A = nº de meses desde la fecha de facturación en que el equipo es utilizado
B = periodo de depreciación (60 meses)
C = coste del equipo (sin IVA)
D = % del uso que se dedica al proyecto (habitualmente 100%)

SUBCONTRATACIÓN DE TAREAS

Descripción	Empresa	Coste imputable
	Total	0,00

OTROS COSTES DIRECTOS DEL PROYECTO[e]

Descripción	Empresa	Costes imputable
Energía Electrica	Universidad Carlos III de Madrid	63.582,56
	Total	63.582,56

[e] Este capítulo de gastos incluye todos los gastos no contemplados en los conceptos anteriores, por ejemplo: fungible, viajes y dietas, otros,...

6.- Resumen de costes

Presupuesto Costes Totales	**Presupuesto Costes Totales**
Personal	25.600
Amortización	63.855
Subcontratación de tareas	0
Costes de funcionamiento	63.583
Costes Indirectos	30.607
Total	183.645

Bibliografía

[1] J. L. Volakis, A. Chatterjee, and L. C. Kempel, *Finite Element Method for Electromagnetics*, ser. IEEE/OUP Series on Electromagnetic Wave Theory. IEEE Press, 1998.

[2] G. de Radiofrecuencia UC3M. (2012) Universidad Carlos III, wiki del departamento Teoría de la Señal y las Comunicaciones. [Online]. Available: http://ls6.tsc.uc3m.es/wiki/index.php/Portada

[3] I. Martinez-Fernandez, "Creación y Validación de un Cluster de Computación Científica Basado en Rocks," Jul. 2009, Proyecto Fin de Carrera. Ingeniería de Telecomunicación, Escuela politécnica superior de la Universidad Carlos III de Madrid.

[4] (2012) Condor project website. [Online]. Available: http://research.cs.wisc.edu/condor/

[5] ANSYS. (2012) Página Oficial de ANSYS HFSS. [Online]. Available: http://www.ansoft.com/products/hf/hfss/

[6] ORACLE. (2012) N1 Grid Engine 6 Administration Guide. [Online]. Available: http://docs.oracle.com/cd/E19080-01/n1.grid.eng6/817-5677/index.html

[7] Llorente-Romano, Gimeno, V. E. Boria-Esbert, and M. Salazar-Palma, "Characterization of resonances by polar expansion of generalized admittance matrix," in *40th*, 2010.

[8] A. Garcia-Lamperez, R. Gomez-Garcia, and M. Salazar-Palma, "Compact diplexer with edge-coupled and nonbianisotropic split-ring resonators," in *2012 IEEE MTT-S International Microwave Symposium Digest*, june 2012.

[9] C. M. L. Sánchez, I. Martinez-Fernandez, and L. E. García-Castillo, "Simulación paramétrica distribuida en hfss." *XXVII Simposium Nacional de la Unión Científica Internacional de Radio, URSI 2012*, 2012.

[10] URSI. (2012) URSI 2012 Website. [Online]. Available: http://www.ursi2012.org/

[11] L. Demkowicz, *Computing with hp Finite Elements. I. One- and Two-Dimensional Elliptic and Maxwell Problems.* Chapman & Hall/CRC Press, Taylor and Francis, 2007.

[12] *Product Documentation: MATLAB*, MathWorks, 2012.

[13] J. J. Merelo. (2012) Geneura Team Website. [Online]. Available: http://geneura.ugr.es/~jmerelo/tutoriales/heuristics101/

[14] P. J. Bentley, *Digital Biology.* Addison-Wesley, 2002.

[15] D. L, *Handbook of Genetic Algorithms.* Van Nostrand Reinhold, 1991.

[16] D. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, 1989.

[17] M. Gestal Pose. (2012) Universidad de Coruña, Dpto. Tecnologías de la Información y las Comunicaciones Website. [Online]. Available: http://sabia.tic.udc.es/mgestal/cv/AAGGtutorial/TutorialAlgoritmosGeneticos.pdf

[18] LIDI. (2012) LIDI Website. [Online]. Available: http://weblidi.info.unlp.edu.ar/catedras/neuronales/09_Tecnicas%20de%20Seleccion.pdf

[19] AWR. (2012) Página Oficial de AWR MWO. [Online]. Available: http://web.awrcorp.com/Usa/Products/Microwave-Office/

[20] F. Alessandri, M. Chiodetti, A. Giugliarelli, D. Maiarelli, G. Martirano, D. Schmitt, L. Vanni, and F. Vitulli., "The electric-field integral-equation method for the analysis and design of a class of rectangular cavity filters loaded by dielectric and metallic cylindrical pucks ." *IEEE Transactions on Microwave Theory and Techniques, VOL. 52, NO. 8*, 2004.