UNIVERSITAT POLITÈCNICA DE VALÈNCIA

UNIVERSIDAD POLITÉCNICA DE VALENCIA

Escuela Técnica Superior de Ingeniería del Diseño

Trabajo Final de Grado

Ingeniería Mecánica

DESARROLLO DE UN ALGORITMO DE CÁLCULO DE SENSIBILIDADES DE FORMA PARA OPTIMIZACIÓN GEOMÉTRICA DE COMPONENTES ESTRUCTURALES CON MALLADOS CARTESIANOS DE ELEMENTOS FINITOS

Autor: David Muñoz Pellicer

Tutor: Juan José Ródenas García

Valencia

Septiembre de 2016

AGRADECIMIENTOS

A mi familia, por apoyarme siempre.

A mi tutor, Juanjo, por estar cuando lo he necesitado y haberme dado un punto de vista interesante a la hora de afrontar los problemas que iban surgiendo.

A Onofre, por ayudarme cada vez que me estancaba en algo, por haberme enseñado muchas cosas que me han sido de gran utilidad y por haber parado de trabajar hasta que mi problema estuviera solucionado.

Al Departamento de Ingeniería Mecánica y de Materiales, por dejarme un hueco en el que poder trabajar y cederme los equipos que he necesitado.

A todos vosotros, os doy las gracias.

RESUMEN

En este proyecto se pretende implementar un optimizador de forma de componentes estructurales sobre el software de elementos finitos, llamado FEAVox, desarrollado en el Departamento de Ingeniería Mecánica y Materiales de esta universidad. Este optimizador será el encargado de encontrar una geometría con volumen mínimo y que cumpla unas restricciones, normalmente referentes a la tensión máxima admisible, a partir de una geometría parametrizada del componente.

Con el fin de que sea más eficaz se ha implementado el cálculo de sensibilidades de forma para obtener los gradientes de la solución con respecto a las variables de diseño. Estos gradientes serán usados por el optimizador para guiar el proceso de optimización.

Por último, se han querido probar algunas mejoras que se centren en reducir el tiempo de ejecución del programa. Por un lado, el optimizador será capaz de buscar geometrías ya calculadas y simplemente recuperar los resultados en lugar de volverla a calcularlos. Por otro lado, se ha añadido un módulo de redes neuronales artificiales que, tras el adecuado proceso de entrenamiento de la red neuronal, permitirá generar inmediatamente los resultados sin necesidad de análisis completos de elementos finitos.

RESUM

En aquest projecte es pretén implementar un optimizador de forma de components estructurals sobre el programari d'elements finits, anomenat FEAVox, desenvolupat en el Departament d'Enginyeria Mecànica i Materials d'aquesta universitat. Aquest optimizador serà l'encarregat de trobar una geometria amb volum mínim i que complisca unes restriccions, normalment referents a la tensió màxima admissible, a partir d'una geometria parametritzada del component.

Amb la finalitat de que siga més eficaç s'ha implementat el càlcul de sensibilitats de forma per a obtenir els gradients de la solució pel que fa a les variables de disseny. Aquests gradients seran usats pel optimizador per a guiar el procés d'optimització.

Finalment, s'han volgut provar algunes millores que se centren a reduir el temps d'execució del programa. D'una banda, el optimizador serà capaç de cercar geometries ja calculades i simplement recuperar els resultats en lloc de tornar-la a calcular-los. D'altra banda, s'ha afegit un mòdul de xarxes neuronals artificials que, després de l'adequat procés d'entrenament de la xarxa neuronal, permetrà generar immediatament els resultats sense necessitat d'anàlisis complet d'elements finits.

ABSTRACT

This project aims to implement a shape optimizer of structural components on the finite element software, called FEAVox, developed in the Department of Mechanical Engineering and Materials of this university. This optimizer will be responsible for finding a minimum volume and geometry that complies some restrictions, usually concerning the maximum permissible stress from a parameterized geometry of the component.

In order to make it more effective we have implemented a shape sensitivities calculating as a way to obtain gradients of the solution with respect to the design variables. The optimizer will use these gradients to guide the optimization process

Finally, we have tried some improvements focused on reducing the execution time of the program. On the one hand, the optimizer will be able to search in the results of calculated geometries and simply retrieve the results rather than re-calculate them. On the other hand, it has been added a module of artificial neural networks. With the adequate training process of the neural network, they will immediately generate results without the need of a complete finite element analysis.

ÍNDICE

Memoria

1 INTRODUCCIÓN

El objetivo del proyecto es el desarrollo de un optimizador de forma para componentes estructurales, basado en:

a) el cálculo de las sensibilidades de forma (derivadas de la solución con respecto a las variables que definen la forma del componente) de las diferentes geometrías, y
b) el uso (entrenamiento y ejecución) de redes neuronales artificiales que tendrían la función de reducir el número de análisis numéricos completos a realizar durante la optimización.

La optimización es una técnica matemática que se utiliza para obtener el punto óptimo de diferentes tipos de problemas. Aplicado al caso que nos concierne, la optimización de forma busca el contorno que minimice ciertos parámetros, dependientes de las variables que definen dicho contorno, satisfaciendo una serie de restricciones impuestas.

En el ámbito industrial esto se traduce, por ejemplo, en la obtención de piezas con un peso o volumen mínimo, pero garantizándose que no se supera la tensión de fallo del material, o que la deflexión máxima está por debajo de los valores límite especificados. Estos procedimientos permiten, por tanto, desarrollar componentes estructurales con prestaciones mejoradas. La importancia de estos procedimientos es evidente en sectores como el aeroespacial donde la reducción del peso supone grandes mejoras de eficiencia, o en el de la automoción, debido a la producción de grandes series de componentes.

El optimizador desarrollado en este proyecto se sirve de algoritmos basados en el gradiente para obtener la configuración óptima. Sea x el conjunto de variables que definen la geometría del componente. Estos algoritmos buscan el valor de x que genere el mínimo de una función $f(x)$, denominada función objetivo, siendo las direcciones de búsqueda las definidas por el gradiente de la propia función para cada valor de x. El gradiente es simplemente la derivada direccional de la función, de manera que, en general, los algoritmos de optimización tenderán a buscar el óptimo siguiendo las direcciones de máximo gradiente.

Se podría decir que el proceso de optimización está divido en dos partes bien diferenciadas (ver *Figura 1*). Por un lado, estaría el llamado **nivel superior** que haría referencia al algoritmo de optimización en sí, que es el encargado de proponer las configuraciones geométricas a analizar y procesar los datos de los análisis para proponer nuevas configuraciones. Y, por otro lado, estaría el llamado **nivel inferior**, gobernado por el método numérico encargado de obtener los resultados de las geometrías propuestas. En problemas de contorno (gobernados por una ecuación diferencial en el interior del volumen a analizar, sujetos a una serie de condiciones de contorno) es común usar el método de los elementos finitos (MEF o FEM por sus siglas en inglés) como método numérico para determinar los resultados del análisis de cada geometría: en cuanto a la función objetivo, la masa o el volumen de la geometría y, en cuanto al cumplimiento de las restricciones, la tensión máxima, por ejemplo. Estos resultados han de ir acompañados de una precisión mínima para tomarlos como buenos.

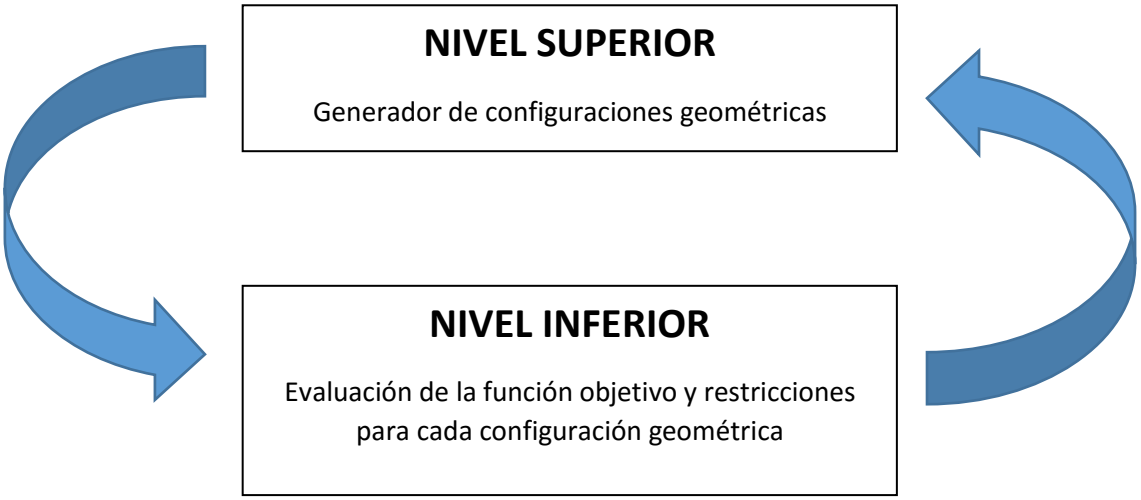

Figura 1. Niveles Superior e Inferior del proceso de optimización.

Debido a que la optimización estructural es un proceso iterativo, se realizan gran cantidad de análisis de geometrías diferentes. Como se ha comentado anteriormente, estas geometrías se analizan mediante el MEF, lo cual conlleva un elevado coste computacional. Por tanto, es el nivel inferior el responsable de que el tiempo de optimización de se eleve mucho.

En este proyecto también se han desarrollado varios métodos para reducir este coste computacional. Entre estos métodos destaca una programación eficiente y la aplicación de redes neuronales, que proporcionan de manera casi inmediata una aproximación a los resultados del análisis, para limitar el número de cálculos mediante el MEF. Por lo tanto, el nivel inferior se amplía al añadir las redes neuronales al método numérico como forma de obtener resultados.

1.1 MARCO TEÓRICO

En este apartado se detallará de una manera más amplia los conceptos vistos en la introducción y se procederá a explicar las herramientas usadas para el desarrollo de este proyecto.

1.1.1 Método de los elementos finitos

El MEF, que obtiene su nombre debido a que el problema definido en un dominio se divide en subdominios (elementos finitos) definidos por los nodos que los conectan, es una de las técnicas más usadas para la resolución de los problemas de contorno que se encuentran en el ámbito de la ingeniería. Un problema de contorno es aquel que está gobernado por ecuaciones diferenciales o integrales dentro de un dominio y por las propias condiciones de contorno en la frontera del mismo. Como se ha comentado, el MEF es un método numérico mediante el cual se realiza una aproximación de las soluciones de las ecuaciones diferenciales. Esto conlleva a la necesidad de cuantificar y controlar el error numérico de los resultados, dado que errores excesivos serían muy perjudiciales para la evolución del proceso de optimización.

1.1.2 Matlab

Matlab es una herramienta de software matemático que ofrece un entorno de desarrollo integrado y un lenguaje de programación propio. Es una herramienta fácil de aprender, a pesar de la inmensidad de sus posibilidades, gracias ser un lenguaje de programación sencillo, a que está muy extendida y a que los usuarios ofrecen mucha ayuda ante posibles problemas que puedan aparecer.

La elección de esta herramienta para implementar el programa se basa en la facilidad para la manipulación de matrices, así como las características que ofrece para gestionar las funciones de optimización gracias a su toolbox de optimización y para la creación de redes neuronales artificiales. También se basa en que el software de elementos finitos, que se usa como base de este trabajo, ha sido desarrollado en Matlab en el Área de Ingeniería Mecánica de la Universitat Politècnica de València donde se ha desarrollado el trabajo.

1.1.3 GiD

GiD es un pre y pos procesador universal, adaptativo y amigable para simulaciones numéricas en ciencia e ingeniería. En este proyecto ha sido usado para la generación de geometrías.

1.1.4 Redes Neuronales

Las redes neuronales artificiales (RNA, o ANN por sus siglas en inglés, o también NNet de Neural Networks) son un intento de simular, al menos parcialmente, la estructura y las funciones de cerebros y sistemas nerviosos de seres vivos. Sin entrar en detalles, una RNA es un sistema de procesamiento de información compuesto por una gran cantidad de elementos de procesamiento más simples, llamadas neuronas artificiales o simplemente nodos, interconectadas entre sí por conexiones directas, que cooperan para llevar a cabo un procesamiento distribuido en paralelo con el fin de resolver la tarea computacional deseada.

Resumiendo lo anterior, las redes neuronales se encargan de generar una salida a partir de una o varias entradas, como puede verse en la Figura siguiente (*ver Figura 2)*:

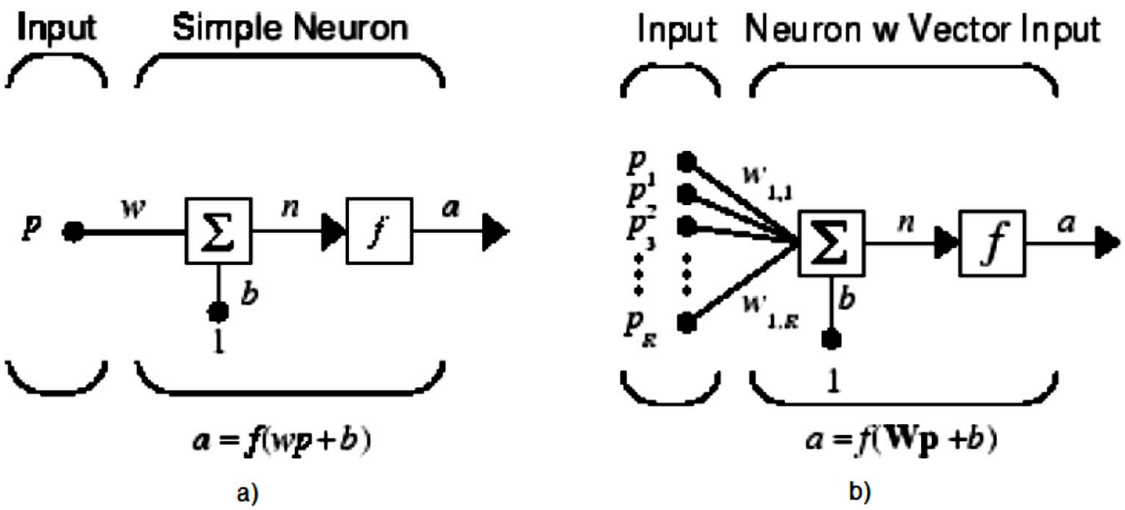

Figura 2. Modelo escalar a) y vectorial b) de una neurona.

Como también puede verse en la Figura, aparece una función de transferencia para modificar la entrada y generar la salida. Esta función de transferencia puede ser de cualquier tipo, pero las más ampliamente utilizadas en ciencia e ingeniería son la lineal, la log-sigmoide y la tan-sigmoide de la Figura 3.

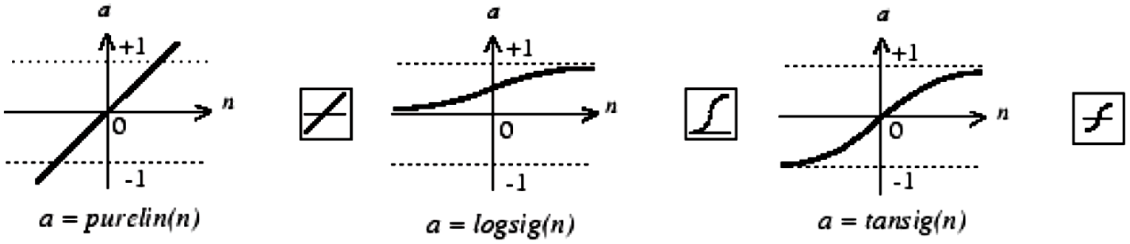

Figura 3. Funciones de transferencia comunes.

Una vez vistas las neuronas simples, vamos a ver como se organizan entre sí para formar redes neuronales. El primer tipo de topología que cabría explicar es aquel en el que varias neuronas se pueden combinar en una misma capa (ver Figura 4):

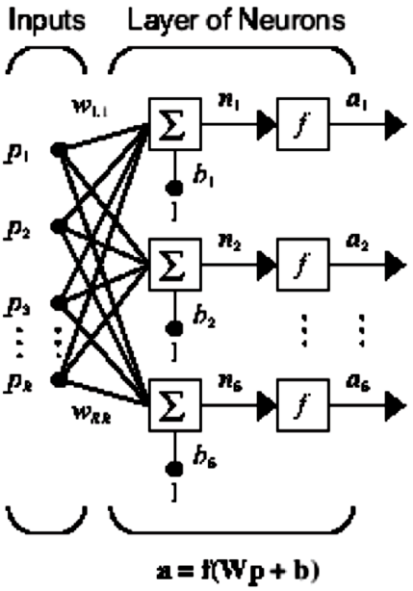

Figura 4. Modelo de red neuronal con una capa de neuronas

En lo referente al segundo tipo de topología, dentro de una red neuronal pueden aparecer varias capas de neuronas *(ver Figura 5):*

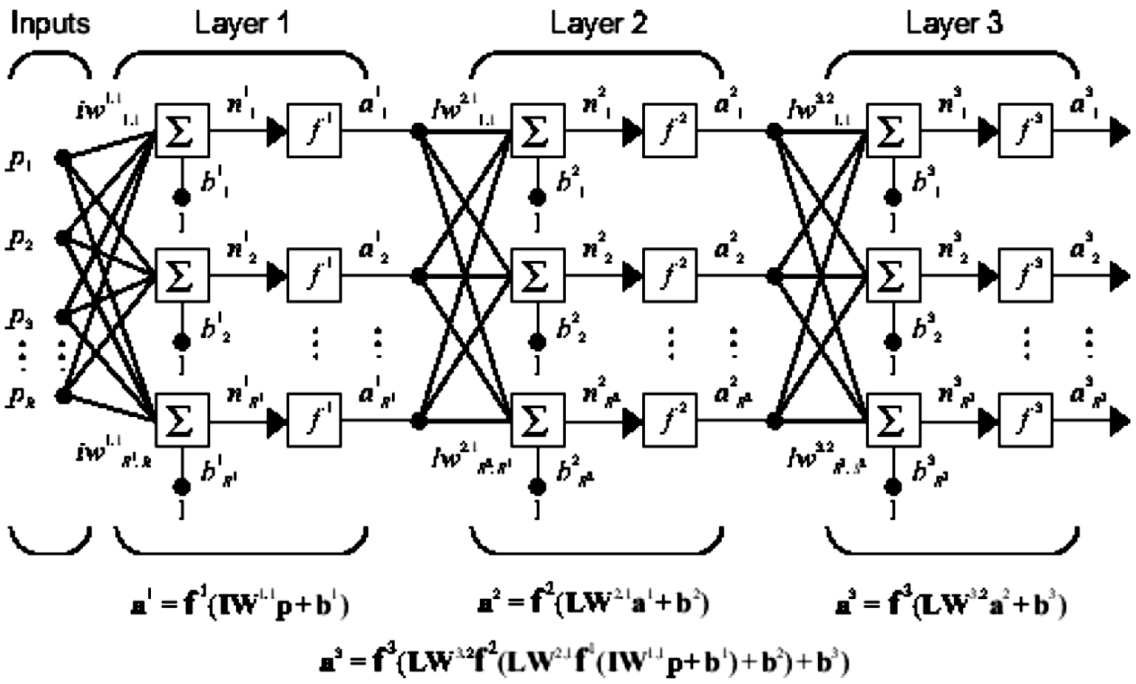

Figura 5. Modelo de red neuronal con múltiples capas de neuronas.

En las RNAs se han de ajustar los parámetros que definen cada una de sus funciones de transferencia para que proporcionen los valores deseados.

El flujo de trabajo para usar correctamente una red neuronal artificial debidamente es el siguiente (*ver Figura 6*):

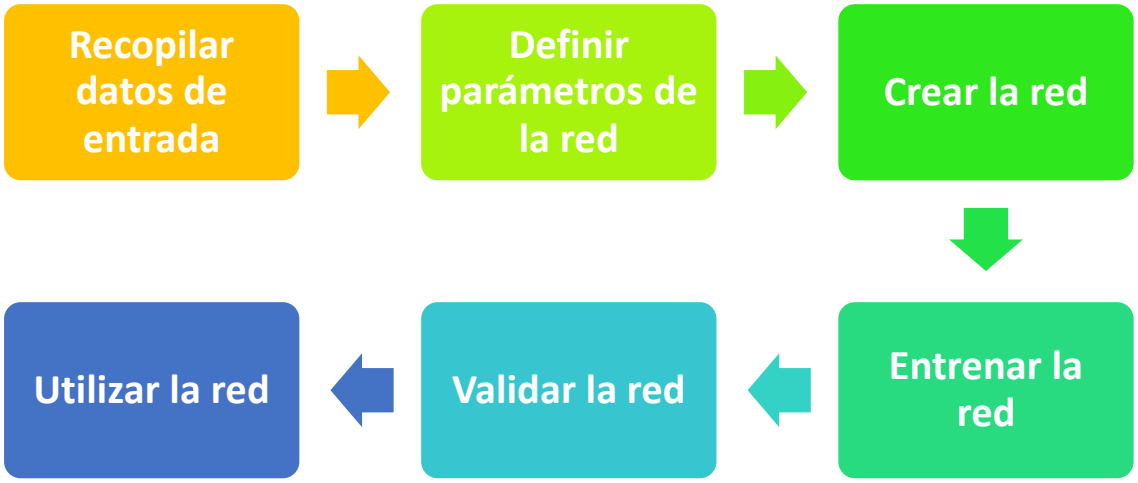

Figura 6. Flujo de trabajo de una red neuronal.

1.1.5 FEAVox

FEAVox es el software de elementos finitos desarrollado en el Departamento de Ingeniería Mecánica y de Materiales. En el proyecto es el encargado de nivel inferior que mencionábamos anteriormente. Es un software, desarrollado en Matlab, que se basa en el uso de mallados Cartesianos independientes a la geometría, a diferencia de la mayoría del software comercial existente a día de hoy, donde la malla se ajusta al contorno del componente a analizar.

1.1.6 Sensibilidades de forma

En el diseño óptimo estructural, el análisis de sensibilidades es el cálculo de las derivadas (gradiente) de la respuesta estructural (desplazamientos, tensiones, etc.) respecto a las variables de diseño. En este proyecto, las sensibilidades de forma van a definir la dirección que debe seguir el optimizador a la hora de generar las combinaciones de valores de las variables de diseño.

1.2 OBJETIVO

Una vez definido el marco teórico en el que se va a situar el proyecto, se puede proceder a definir cuáles son los objetivos a tratar.

El objetivo principal es el de desarrollar un optimizador de forma para componentes estructurales 3D con el fin de que pueda ser adaptado al software ya existente en el departamento y de esa forma mejorar las características de éste, y que pueda ser competitivo ante el software comercial.

Por otro lado, aparecen varios objetivos secundarios, como estudiar las bondades del uso de las redes neuronales como medio para predecir un conjunto de salidas ante ciertos datos de entrada, comparando los resultados obtenidos por este método con los que se obtendrían por medio del software de elementos finitos. Otro objetivo secundario sería el de validar el procedimiento de obtención de las sensibilidades de forma comparándolas con métodos más simples para obtener el gradiente deseado como, por ejemplo, mediante diferencias finitas.

2 DESARROLLO

Este capítulo se centra en dar una descripción mucho más detallada de lo que es un método de optimización, las diferentes técnicas existentes a día de hoy, cómo se aplicarán éstas al optimizador de forma y las herramientas usadas para ello.

Desde sus orígenes, la ciencia y la ingeniería han buscado la forma de obtener la solución óptima de cada uno de los problemas a los que se han enfrentado. Esta búsqueda se ha visto muy limitada a causa de la alta complejidad de los cálculos que es necesario llevar a cabo. Con la aparición del cálculo computacional esta tarea se ha vuelto más sencilla y, gracias a este avance, hoy se pueden desarrollar procedimientos como el que se trata aquí: un optimizador de forma de componentes estructurales.

El problema general que debe abordar un optimizador es el de encontrar los valores de las variables de diseño que definen las condiciones del problema que se correspondan con los mínimos de unas funciones objetivo dadas, pero bajo unas restricciones determinadas y previamente definidas por el usuario.

La principal clasificación de los problemas de optimización se basa en el número de funciones objetivo a minimizar u maximizar. Por un lado, estaría la optimización mono-objetivo en la que solo habría una función objetivo y, por otro lado, tendríamos la optimización multi-objetivo en la que aparecer varias funciones objetivo a optimizar. En este caso abordaremos la optimización mono-objetivo.

Los algoritmos de optimización se pueden clasificar también según los siguientes criterios:

- Algoritmos **deterministas**, **que** producen los mismos resultados finales si se ejecutan desde un mismo punto inicial o algoritmos **estocásticos, que** pueden generar distintos resultados aun siendo iniciados en el mismo punto.
- Algoritmos **basados en el gradiente**, que necesitan la información del gradiente de la función objetivo para definir la dirección a seguir o algoritmos **no basados en el gradiente**, que simplemente necesitan el valor de la función objetivo para realizar la optimización.
- Algoritmos de **solución única**, en los que el próximo punto a analizar solo viene definido por el punto anterior o algoritmos de **solución múltiple** en los que la elección del futuro punto a analizar ha sido definida por un conjunto de puntos anteriores.

Algunos ejemplos de los algoritmos **deterministas** más ampliamente utilizados:

- No basados en el gradiente:
    1. Métodos de ascenso de colinas.
    2. Método Simplex.
    3. Método coordinado.
- Basado en el gradiente:
    1. Método del gradiente conjugado, necesita la 1ª derivada.
    2. Métodos de Quasi-Newton, necesita la 1ª derivada.
    3. Método de Newton, necesita la 2ª derivada.
    4. Método de Levenberg-Marquardt, necesita la 2ª derivada.

Como se ha visto anteriormente, los algoritmos deterministas generan el mismo resultado en las diferentes ejecuciones del problema si se parte del mismo punto. Esto representa un gran beneficio, sobre todo para la persona encargada de estudiar cómo evoluciona este proceso, ya que puede modificar algunos de los parámetros que definen el problema y saber cómo debería ser el resultado final. Otro de los grandes beneficios que nos aporta este tipo de algoritmos es una convergencia habitualmente más rápida comparada con los algoritmos estocásticos. Algo a tener en cuenta es que todos los algoritmos anteriormente mencionados convergerán hacía el primer mínimo que se encuentren en su camino, el cual puede ser un mínimo local en vez del mínimo global, que sería el que interesaría obtener. Para evitar que la solución sea un mínimo local debemos servirnos de alguna herramienta que ejecute estos algoritmos desde diferentes puntos con el fin de abarcar el mayor número de puntos y asegurar la convergencia hacia el mínimo global. Evidentemente, esto aumentará el coste computacional global del proceso.

En el software desarrollado se usará un algoritmo determinista basado en el gradiente, ya que gracias a este se puede definir la dirección de búsqueda a seguir más adecuada, lo cual suele traducirse a una convergencia más veloz. Dentro de los algoritmos de este tipo vistos anteriormente se utilizará el método de Quasi-Newton, ya que el software de elementos finitos del departamento tiene un módulo encargado de obtener las sensibilidades de forma que puede ser usada como la primera derivada de la función objetivo y de las restricciones respecto a las variables de diseño. Además de esta derivada, obtenida mediante el análisis de sensibilidades, se ha implementado inicialmente un procedimiento de obtención de derivadas basado en el método de las diferencias finitas. Aunque es menos preciso, este procedimiento resulta más sencillo de implementar que el cálculo de sensibilidades, con lo que puede ser usado para validar la implementación del análisis de sensibilidades.

Una vez se ha visto el fundamento teórico que define un problema de optimización, se verá cuáles son las herramientas y cómo se usan éstas para implementar lo visto anteriormente.

El toolbox de Optimización es un módulo de Matlab en el que se han recogido un gran número de funciones de optimización existentes. De esta forma se amplía la capacidad de computación numérica de Matlab, a la vez que facilita su uso para los usuarios. Todas estas funciones son ficheros con extensión “.m” implementan los algoritmos ya vistos.

De todos los algoritmos que existen actualmente en este toolbox cabe destacar tres de los más usados. En primer lugar se detallará el algoritmo utilizado y después dos algoritmos ampliamente usados pero que para el proyecto no eran adecuados.

La función encargada de implementar los algoritmos de optimización en Matlab es *fmincon* que resuelve el problema de optimización formulado como sigue (*ver Ecuación 1*):

$$\min_{x} f(x) \text{ such that } \begin{cases} c(x) \leq 0 \\ ceq(x) = 0 \\ A \cdot x \leq b \\ Aeq \cdot x = beq \\ lb \leq x \leq ub, \end{cases}$$

Ecuación 1 *Descripción de fmincon.*

Los parámetros vistos en la Figura corresponden con las siguientes definiciones:

- *f(x),* función objetivo a minimizar.
- c(x), corresponde a las restricciones de desigualdad del problema.
- *ceq(x),* corresponde a las restricciones de igualdad del problema.
- *A y b,* parámetros que representan las restricciones de desigualdad entre las variables de diseño.
- *Aeq* y *beq*, parámetros que representan las restricciones de igualdad entre las variables de diseño.
- *lb y ub,* corresponden con los límites inferior y superior que contienen los valores de las variables de diseño.

Aplicando todos estos parámetros al tipo de problema que se analizará en este proyecto se ve que *f(x)* es el volumen de la pieza a evaluar, *x* hace referencia a las coordenadas de los puntos que definen la geometría del componente a analizar, la restricciones de desigualdad *c(x)* representarían los estados límite que puede soportar la pieza evaluada (por ejemplo, la condición de máxima tensión de Von Misses a fluencia o un desplazamiento máximo admisible), la restricciones de igualdad *ceq(x)* harían la misma función que las anteriores en caso de que las hubiera, la matriz *A* y el vector *b* para desigualdades y *Aeq* y *beq para igualdades* serían los parámetros que representan las restricciones entre las variables de diseño y, por último, los límites inferior *lb* y superior *ub* serían los límites de cada variable de diseño.

Los otros algoritmos que se han considerado como funcionan de una manera similar a *fmincon* y son: *fgoalattain* (*Ecuación 2*) y *fminimax* (*Ecuación 3*) Ambos empleados en problemas multi-objetivo.

$$\underset{x,\gamma}{\text{minimize}}\ \gamma \quad \text{such that} \begin{cases} F(x) - weight \cdot \gamma \leq goal \\ c(x) \leq 0 \\ ceq(x) = 0 \\ A \cdot x \leq b \\ Aeq \cdot x = beq \\ lb \leq x \leq ub. \end{cases}$$

Ecuación 2 *Descripción de fgoalattin.*

$$\min_{x} \max_{i} F_i(x) \quad \text{such that} \begin{cases} c(x) \leq 0 \\ ceq(x) = 0 \\ A \cdot x \leq b \\ Aeq \cdot x = beq \\ lb \leq x \leq ub \end{cases}$$

Ecuación 3 *Descripción de fminimax.*

Estos algoritmos presentan ciertas desventajas frente a *fmincon*:

- Pueden incumplir alguna de las restricciones en alguna de las iteraciones llevadas a cabo en el proceso. Asumen que las funciones objetivo siempre son continuas. Esto se cumplirá en los ejemplos numéricos de este trabajo, pero no se puede asumir desde los primeros pasos de la optimización.
- Pueden converger únicamente en soluciones locales.
- No permiten trabajar con el algoritmo *GlobalSearch*, el cual nos permite encontrar la solución global.
- Particularmente, *fgoalattain* es usado principalmente para casos en los que se busquen optimizar varias funciones y *fminimax* se usa cuando se busca también el máximo de la función, lo cual generará un coste computacional innecesario al no ser lo que buscamos.

Por todas estas características y por las ventajas que presenta el uso de *fmincon* se han descartado *fgoalattain* y *fminimax* como posibles algoritmos encargados de gobernar el proceso de optimización.

¿Por qué es importante el uso de *GlobalSearch?* Este algoritmo permite encontrar la solución global óptima haciendo un barrido de las posibles soluciones, iniciando el algoritmo *fmincon* desde diversos puntos. Con este algoritmo se tiene la posibilidad de elegir la cantidad de puntos de inicio potenciales (*NumTrialPoints*) además del punto sugerido por el usuario como *x0*. Estos puntos deben pasar unas pruebas antes de ser considerados válidos, si no son descartados. Otra

propiedad a tener en cuenta es que permite elegir cuántos de los posibles puntos a evaluar son examinados en la primera etapa del proceso (*NumStageOnePoints*). La última de las propiedades interesantes es que se puede discriminar los puntos de inicio a analizar a través de una clasificación propuesta por el algoritmo (*StartPointsToRun*).

El proceso que sigue *GlobalSearch* se mostrará en el siguiente diagrama de flujo, explicando los pasos a seguir (*ver Figura 7*):

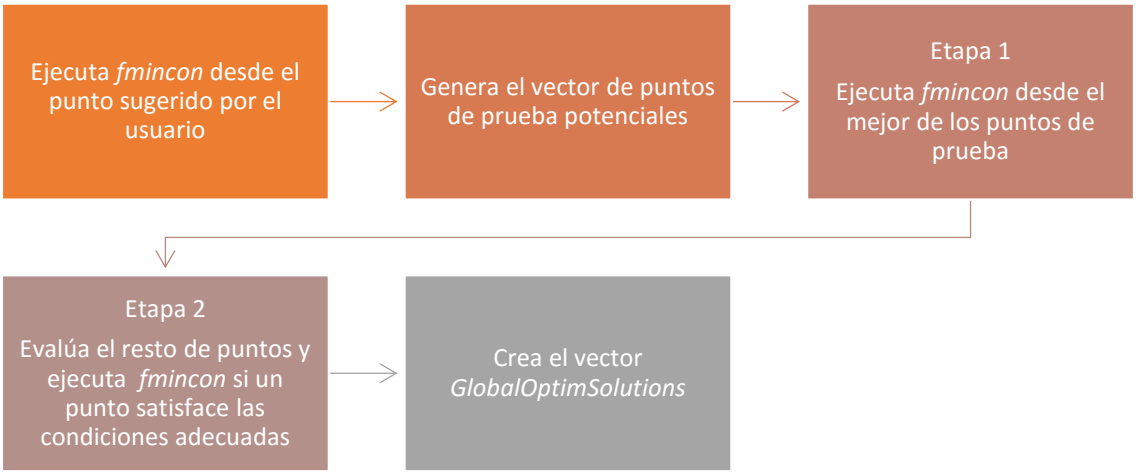

Figura 7. Diagrama de proceso de GlobalSearch.

También existen otros algoritmos especialmente diseñados para iniciar funciones de optimización desde diversos puntos con el fin de abarcar el mayor rango de muestreo posible, como puede ser el algoritmo llamado *MultiStart* el cual posee ciertas diferencias respecto a *GlobalSearch* (*ver Figura 8*):

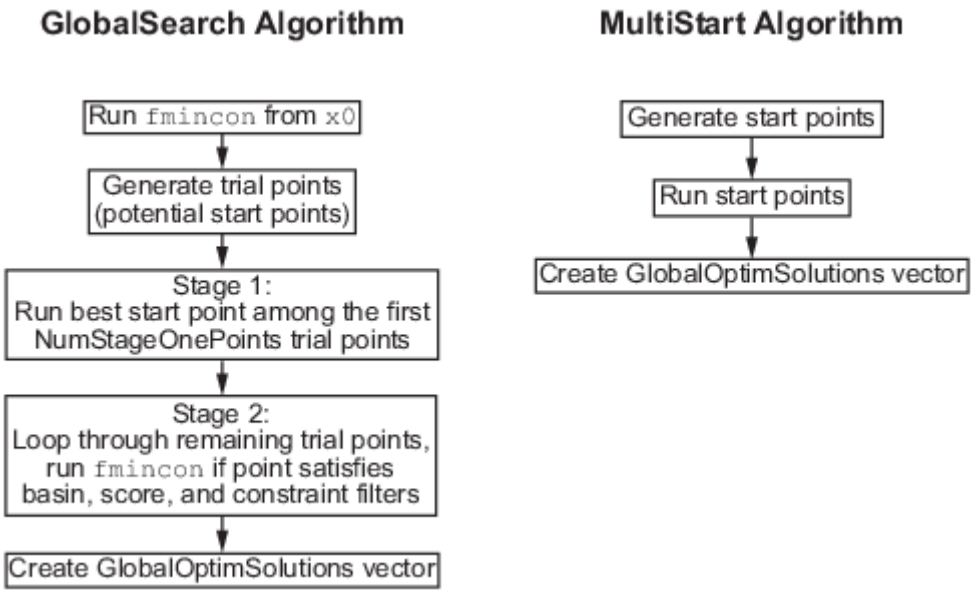

Figura 8. Comparación entre la ejecución de GlobalSearch y de MultiStart.

Aparte de las diferencias claras en los pasos que siguen cada uno de estos algoritmos para generar una solución, también existen otras. Mientras que *GlobalSearch* utiliza un método de búsqueda por dispersión (más efectivo), *MultiStart* busca los nuevos puntos a analizar a través de un sistema de distribución uniforme. *GlobalSearch* analiza los puntos de inicio y rechaza aquellos que no cumplen con unas condiciones necesarias, al contrario que *MultiStart* que ejecuta todos los puntos de inicio. Por último, *GlobalSearch* es el único que puede ser usado con *fmincon* mientras que *MultiStart* puede usar los otros algoritmos vistos anteriormente.

Resumiendo, *GlobalSearch* es mucho más efectivo en la búsqueda de un único mínimo global a través de un solo algoritmo de optimización, mientras que *MultiStart* se usa más a menudo para optimizar varias funciones objetivo simultáneamente, permitiendo el uso de la ejecución en paralelo, lanzando varios algoritmos al mismo tiempo. En nuestro caso se usará *GlobalSearch* puesto que es el único con el que se puede usar *fmincon.*

Todas las propiedades necesarias para el correcto funcionamiento de *GlobalSearch* se definirán y posteriormente se almacenarán en una variable denominada *gs*.

Por otro lado, las opciones que regirán el proceso de optimización quedarán fijadas en la variable *opts* gracias a la instrucción *optimset*. Entra esas opciones se encuentran el algoritmo interno que debe seguir *fmincon* para encontrar los mínimos de la función objetivo (*interior-point, sqp, active-set o truste-region-reflective)*, se puede seleccionar también el uso o no de los gradientes de la función y restricciones proporcionados por el usuario a través de diferencias finitas o del cálculo de sensibilidades (*GradObj* y *GradConstr*) y por último las tolerancias de las variables que entran en juego en la optimización, como son la variable de diseño, la función objetivo y las restricciones (*TolX, TolFun,* y *TolCon*).

La última de las variables que definen el problema sería *problema* y en ella se guardaría la estructura del proceso de optimización gracias a una instrucción llamada *createOptimProblem.* En esta variable quedaría reflejado lo siguiente:

- El solver o algoritmo usado, que en el caso de trabajar con *GlobalSearch* estaría limitado únicamente a *fmincon*.
- La función objetivo a minimizar, en este caso el volumen de la geometría.
- Las funciones que definen las restricciones, aquí la forma de la Figura estaría limitada por una tensión máxima admisible únicamente, pero se podría restringir también a través de unos desplazamientos dados.
- Las restricciones de igualdad y desigualdad que relacionan las variables de diseño.
- Los límites inferior y superior que pueden tomar las variables de diseño.
- Las opciones previamente definidas en la variable *opts*.

Para resumir, *GlobalSearch* permitirá ejecutar *fmincon* un determinado número de veces para barrer el mayor número de puntos posibles y encontrar el mínimo global, por otro lado, la estructura del problema queda definida en la variable *problema* junto con unas opciones determinadas almacenadas en *opts* y por último, todo el problema se lanzará mediante una instrucción apropiada: *run(gs,problem)*.

3 DISEÑO

Una vez ya han sido detalladas las funciones que gobernarán el proceso de optimización, se ha de explicar el funcionamiento de la parte restante del código que se encarga de obtener los resultados de la función objetivo y restricciones, así como de organizar todo el proceso y de almacenar parámetros necesarios.

3.1 FUNCIONAMIENTO DEL PROGRAMA

El flujo que sigue la información dentro del programa es el siguiente, su representación gráfica se muestra en la Figura 9:

1. El usuario define todos los parámetros de la optimización mediante la función *Optimization*, que es la encargada de gobernar todo el programa. El almacenaje de datos se hace en segundo plano mediante la función *OptimParameters*. En el script *Optimization* también se pueden definir muchas variables referentes al cálculo MEF como, por ejemplo, el tamaño de la malla o, incluso, del análisis de sensibilidades.
2. *Optimization* llamará a *OptimFunction*, script que contiene la llamada al optimizador en sí, pudiendo elegir la posibilidad de lanzarlo completo con *GlobalSearch* o simplemente un *fmincon* que concluirá en el primer mínimo local.
3. El solver elegido será el encargado de evaluar la función objetivo llamando a la función *ObjectiveFun* o las restricciones llamando a la función *RestrictionFun*.
4. Dentro de ambas rutinas existe una parte común almacenada en *OutputObtention* y una parte que es particular para cada una, ya que en cada una se buscan resultados diferentes.
5. El script *OutputObtention* es el encargado de subdividir el método mediante el cual se obtiene el resultado deseado.
    a. Por un lado, se puede obtener el resultado a partir de memoria (*MemoryResults*), es decir, recuperando resultados si ese punto en particular ya se ha calculado previamente y, por tanto, se encuentra almacenado en la matriz *ResultData*, donde se guarda todo tipo de información útil de cada una de las geometrías analizadas.
    b. Por otro, estaría el módulo de redes neuronales que es el encargado de crear las redes (*NNetCreation*) y después de esto, usarlas para obtener resultados (*NNetResults*). En este módulo los gradientes se obtienen a partir de diferencias finitas, puesto que este método ya es una aproximación, no se pierde demasiada precisión al asumir esto.
    c. Por último, está la parte en la que se usa FEM para obtener resultados (*FEMResults*).
6. El primer paso en el módulo *FEMResults* es la creación de la geometría y para ello se ha implementado la siguiente función, Prob_*GeometryGenerator*. Dentro de esta función existen dos formas para generar la geometría, si la geometría solo depende de una variable, la geometría se puede generar directamente en *Matlab* mediante la creación de rectas y arcos a partir de los puntos conocidos y también se puede generar a través de *GiD* en segundo plano. Para el caso de más de una variable la geometría solo se podrá ser creada a partir de *GiD*.

7. Con la geometría ya generada, se introduce en *FEAVox* y éste nos devuelve los resultados que buscábamos, volumen y tensión principalmente. Para obtener el gradiente de estas funciones se usa diferencias finitas si la malla es muy basta y en caso contrario actuará el módulo de sensibilidades. En caso de que *FEAVox* de error se intentará aproximar las soluciones si existe un punto almacenado muy cercano a través de sus gradientes, aunque si esto no fuera posible se devolvería el valor *NaN*.

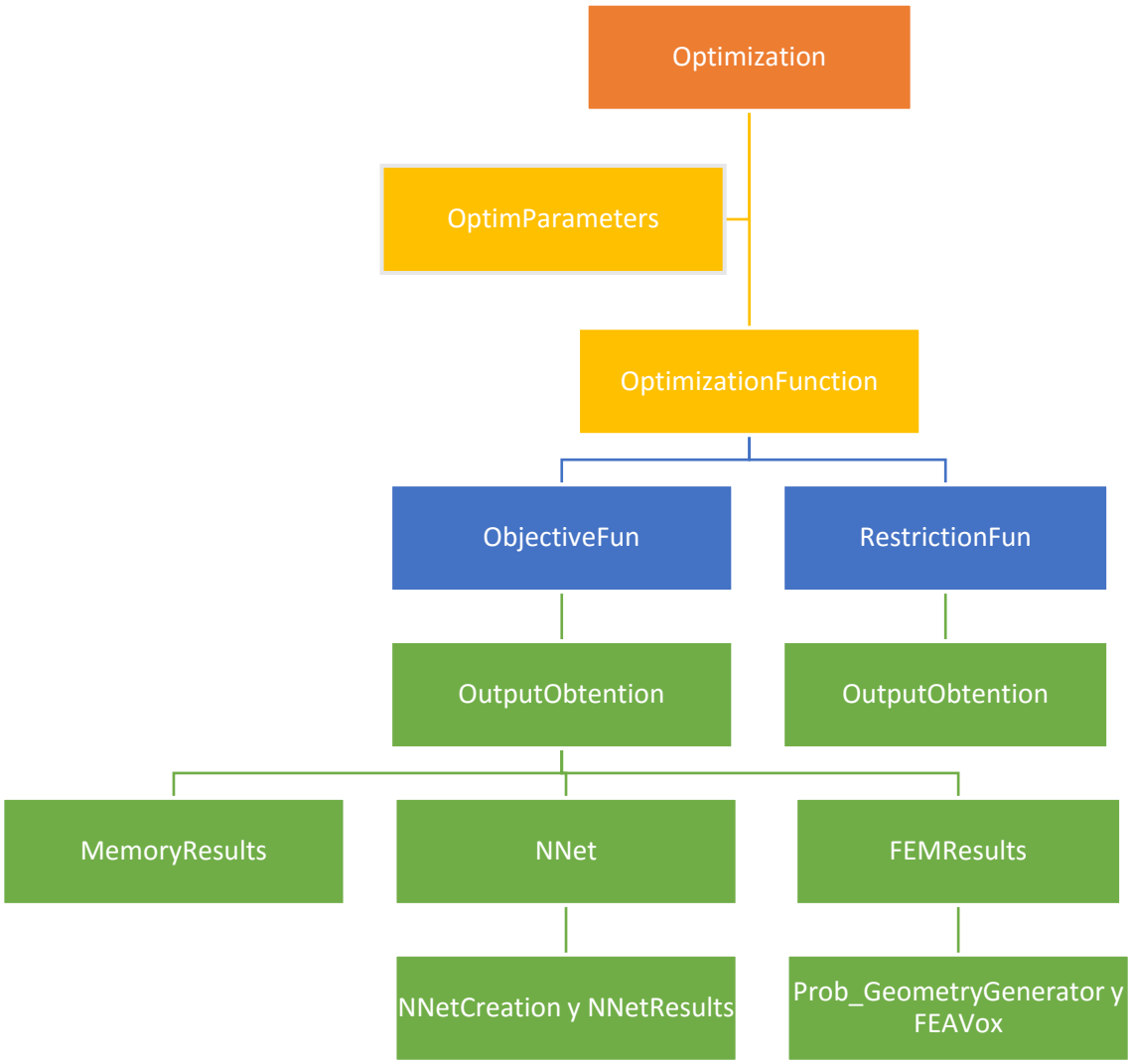

Figura 9. Flujo básico del programa.

Este sería el proceso que sigue programa dentro de una única iteración. Como se ha comentado, este es un proceso iterativo en el que necesita retroalimentación que cierra el proceso. El propio optimizador es capaz de dar esa retroalimentación analizando los resultados obtenidos. Si considera que es necesaria una nueva iteración, usará los resultados de las funciones junto con sus gradientes para seleccionar un nuevo punto de prueba y se ejecutarán las funciones *ObjectiveFun* y *RestrictionFun* hasta que el paso entre dos puntos de prueba (la diferencia entre dos geometrías consecutivas analizadas) sea inferior a un valor muy pequeño *TolX* definido por el analista (*ver Ecuación 4*), es decir:

$$x_i - x_{i-1} < TolX$$

Ecuación 4.

La Figura 10 muestra el esquema resumido del funcionamiento del ciclo iterativo de optimización.

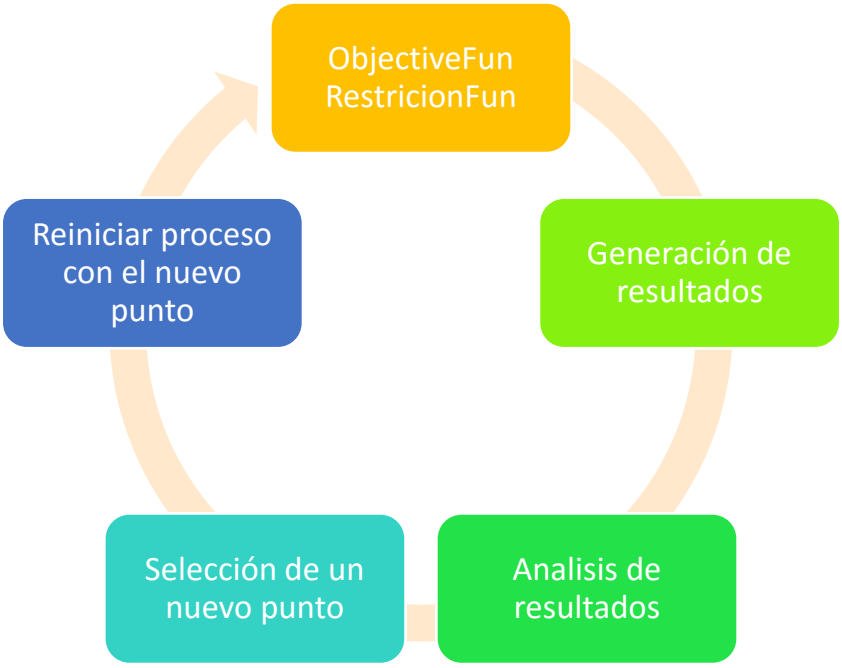

Figura 10. Proceso iterativo.

3.2 CONTENIDO DEL PROGRAMA

El programa de optimizador de forma en 3D está formado por gran cantidad de funciones y módulos como se ha podido ver en la breve descripción del funcionamiento del programa. A continuación, se detallará cada una de estas funciones con el fin de obtener una visión más global de todo el proceso en conjunto.

Por su tamaño y responsabilidad, el programa debería ser partido en tres grandes secciones: la encargada de gobernar todo el proceso (*Optimization*.m [1]), la encargada de generar las soluciones de la función objetivo y restricción (*ObjectiveFun*.m y *RestrictionFun*.m) y por último la función encargada de generar las distintas geometrías (*Prob_GeometryGenerator*.m).

3.2.1 *Optimization.m*

Es la función encargada de gobernar todas las funciones y scripts vistos anteriormente. En él se definen todos los parámetros que el cálculo de FEM seguirá para obtener los resultados. Estos parámetros se definen en unas funciones que forman parte del código de *FEAVox* dejando aparte la propia ejecución del cálculo que aparecerá más adelante. Se puede separar porque esta parte del código se mantendrá igual durante todas las ejecuciones del programa. Así se consigue un código más eficaz y claro. Al igual que ocurre con los parámetros de elementos finitos, también se pueden definir aquí los propios del módulo de sensibilidades, apareciendo de nuevo el código de ejecución más adelante.

Dentro del *script Optimization*.*m*, se ha creado la función *OptimParameters.m*, en ella se definen una serie de parámetros de gran importancia para el correcto funcionamiento de los algoritmos de optimización, *GlobalSearch* y *fmincon*. Para facilitar su uso, la definición de estos valores se realizará desde el script *Optimization.m*. De esta manera el usuario no debe acceder a otras funciones y será *OptimParameters.m* la encargada de asignar estos valores a los parámetros correspondientes. Los valores que el usuario podrá variar con total libertad se explicarán más adelante, *ver Anexo 1: Manual de Usuario.*

Entre todos estos parámetros que la función *OptimParameters.*m es capaz de gestionar aparecen algunos que han sido creado específicamente para este problema, que convendría que no se alterarán, como pueden ser las matrices y vectores de las restricciones de las variables de diseño y los límites de estas variables. También aparecen una serie de variables que almacenan cuáles serán las funciones objetivo y de restricción. Posteriormente se introducirá esta información como parámetros a la hora de definir el problema.

Una vez ya se han definido todos los parámetros que gobernarán el proceso se puede proceder a ejecutar al optimizador en sí, habiéndose creado una función llamada *OptimizationFunction.m* para tal labor. Esta función simplemente sirve como un seleccionador de algoritmo en función del valor de entrada que recibe. Se puede introducir la palabra “Local”, para ejecutar el algoritmo *fmincon* sobre el punto de inicio y que converja en el primer mínimo que se encuentre en su camino. También es posible introducir la palabra “Global” que ejecutaría el optimizador completo a través del algoritmo *GlobalSearch*, buscando el mínimo global de la función objetivo.

[1] *Nótese que se ha añadido la terminación “.m”, ésta hace referencia a las funciones y scripts escritos en código de Matlab, lenguaje M.*

Todo este proceso se detalla en la Figura 11:

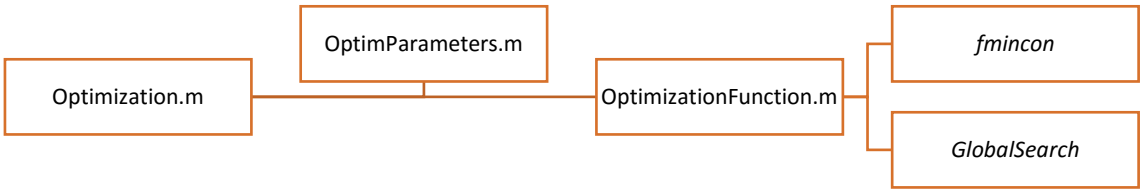

Figura 11. Proceso Optimization.m

3.2.2 ObjectiveFun.m y RestrictionFun.m

Independientemente del algoritmo seleccionado para ejecutar el análisis de optimización, las funciones fundamentales para que este funcione son las dedicadas a obtener los resultados de la función objetivo y de las restricciones, que se detallarán a continuación.

Estas dos funciones se tratan como una ya que generan la misma respuesta, siendo tan similares que se ha extraído todo el código común a ambas y se ha creado un nuevo script llamado *OutputObtention.m*. Para agilizar y flexibilizar el funcionamiento del programa cada una de las funciones será capaz de obtener todos los resultados posibles del análisis MEF, incluidos los resultados del análisis de sensibilidades. Cada función devolverá los resultados para los que ha sido creada y almacenará el resto, esta invención se ha realizado con el fin de ahorrar cálculo de MEF, los cuales son muy costosos.

Estas funciones tienen un solo parámetro de entrada, la variable de diseño, siendo el algoritmo de optimización seleccionado el encargado de suministrarla. En caso de ser un problema de una variable el parámetro será un escalar, pero si tiene más de una la variable de diseño será un vector que almacenará todos los parámetros que definen la geometría del individuo.

Lo primero que se hace en la función es comprobar si el usuario ha seleccionado la opción de cargar un análisis anterior para poder hacer uso de resultados obtenidos previamente, en vez de partir de cero. Si no se selecciona esta característica se creará la variable *ResultData* (*ver Figura 12*). Esta variable es una matriz en la que se guardarán todos los resultados de cada uno de los individuos además de varios indicadores útiles. A continuación, se le asignarán valores nulos a esta matriz. El número de filas de la matriz es función del número de variables de diseño y las columnas almacenarán los resultados de cada una de las geometrías analizadas.

Antes de seguir, se describirá esta variable ya que es fundamental para el programa. En esta matriz se guarda toda la información que pueda ser útil o beneficiosa para el propio optimizador o para el usuario. Un ejemplo de esta matriz sería el siguiente:

Identificador de función	1	2	…	…	Nº Llamadas a cálc.
Identificador de individuo	1	1	…	…	-
Valor de la vble. de diseño	15	15	…	…	-
Volumen	3,13E+03	3,13E+03	…	…	-
Gradiente de volumen	465,18	465,18	…	…	-
Tensión de Von Misses	1,7952	1,7952	…	…	-
Gradiente de tensión	-0,0684	-0,0684	…	…	-
Restricción[2]	-0,2048	-0,2048	…	…	-
Obtención de resultados	1	3	…	…	-
Tiempo de ejecución	3,3641	0,0066	…	…	-

Figura 12. Ejemplo de una variable de diseño de la matriz ResultData.

El número de columna también servirá para conocer el número de llamadas al cálculo de los resultados que realiza el optimizador. En caso de tener más de una variable de diseño esta información se almacenará en columna.

Continuando con la ejecución de la función y con el tamaño de esta variable definido, se empezará a llenar con información, siendo la primera fila la primera a completar y que se indicará con un 1 si ha entrado a *ObjectiveFun*.m o un 2 si los resultados se han obtenido en *RestrictionFun*.m.

Tras esto se procederá a la ejecución de la función *OutputObtention*.m, que no es más que un script en el que se recopilan los diversos métodos para obtener los resultados que el optimizador necesita.

Dentro de *OutputObtention*.m, lo primero que se realiza es la definición de algunas constantes como la máxima tensión admisible en *TensAdm*, el número de análisis mínimo para alimentar las redes en *FeedNNetResults*, la cantidad máxima de usos de la red antes de que se eliminen y se entrenen unas nuevas redes en *MaxNNetUses* y por último, se define un tamaño de malla de los análisis MEF a partir del cual los gradientes se calcularán por sensibilidades en vez de por diferencias finitas (esto se detallará más adelante) que se guardará en la variable *OptimLevel*.

Tras esto, se inicializa el temporizador encargado de medir el tiempo que se tarda en dar un resultado.

[2] *La restricción se refiere a la resta entre la tensión de Von Misses y la tensión máxima admisible en la geometría, se pretende que esta resta sea lo más cercana a 0 posible.*

Ahora comienza la obtención de resultados propiamente dicha, existen tres métodos para obtenerlos: a) recuperar la información de un individuo ya calculado, b) el uso de redes neuronales artificiales y c) mediante el análisis de MEF.

a) En caso de que ya se haya calculado previamente el individuo que se pretende estudiar accederemos a la función *MemoryResults.m.* Ésta se encargará de buscar la posición del individuo ya calculado y recuperará la información para asignársela al nuevo individuo.
b) En caso de que se vayan a usar las redes neuronales, primero se deben de tener la cantidad de datos adecuados (generados por MEF) para poder entrenar la red neuronal. Para dicho entrenamiento se introducirán como parámetros de entrada las variables de diseño y como parámetros de salida, por un lado, el volumen y por otro, la tensión. Se creará una red para cada uno de estos valores. El gradiente en este caso se hallará mediante diferencias finitas, ya que al ser las redes una aproximación no se sacrifica demasiada precisión. Con las redes ya creadas, se introducen los parámetros que definen cada nuevo individuo las redes devuelven los resultados de manera inmediata.
c) En caso de que no se hayan obtenido los resultados por alguno de los métodos anteriores, el individuo será analizado a través de MEF. Generalmente este estudio tendrá dos partes, la generación de la geometría y el cálculo de resultados por elementos finitos de esa geometría.

   Se ejecutará la función *FEMResults.m* con las variables de diseño como entrada, dentro de esta función se generará la geometría y se pasará esa geometría a *FEAVox*, los resultados que generé este código serán los que se pasarán al optimizador además de guardarse en la variable *ResultData*. A continuación, viene el cálculo del gradiente. En función del valor de *OptimLevel* se elegirá un método u otro. En caso de que el tamaño de malla sea menor o igual al valor de *OptimLevel* el gradiente se generará por diferencias finitas, ya que la malla es muy basta y no se puede obtener mucha precisión, mientras que, si es mayor, el gradiente se hallará a partir del cálculo de sensibilidades de forma, ya que para su correcto funcionamiento necesita de mallas más refinadas. El uso de diferencias finitas exige análisis MEF completos con *FEAVox* de varias geometrías: de la de interés y de tantas geometrías como variables de diseño (en cada una de estas geometrías se habrá hecho una variación mínima de una de las variables de diseño con respecto a su valor en la geometría a analizar). Sin embargo, el cálculo de sensibilidades exige una sola ejecución de FEAVox que incluye el análisis MEF de la geometría a analizar y un análisis auxiliar para cada variable de diseño, con el consiguiente ahorro de tiempo de ejecución y aumento de precisión, dado que diferencias finitas representa una aproximación al cálculo de sensibilidades. En caso de que el módulo de sensibilidades falle se ha añadido código para que los gradientes se evalúen mediante diferencias finitas, lo cual es preferible antes de dar al optimizador un error o un valor inaceptable.

   En caso de que elementos finitos falle de una manera global, se ha añadido unas líneas de código mediante las cuáles se obtendrá como resultado el valor *NaN*, lo cual inhabilita ese individuo como posible mínimo. El error es capturado y no hace parar al optimizador. Esto es muy importante ya que son procesos de varias horas y hay que hacer lo posible para que no se produzcan errores que paren el proceso. Aun así, con el fin de no perder estos individuos erróneos se ha añadió código para que a partir de los gradientes de otro individuo muy cercano se puedan interpolar los resultados en estos mediante una aproximación lineal que, aunque induce errores numéricos, podrá considerarse suficientemente precisa si la

geometría a partir de la cual se realiza el cálculo es suficientemente próxima a la que nos interesa.

Estos son los tres métodos elegidos para obtener los resultados de la función objetivo y de la restricción, cuando ya se ha obtenido una salida de las funciones, solo queda parar el temporizador y almacenar este dato en cada uno de los individuos con el fin de comparar cuales son los métodos más rápidos.

El flujo que sigue la información en estas funciones se verá detallado en el diagrama de la Figura 13:

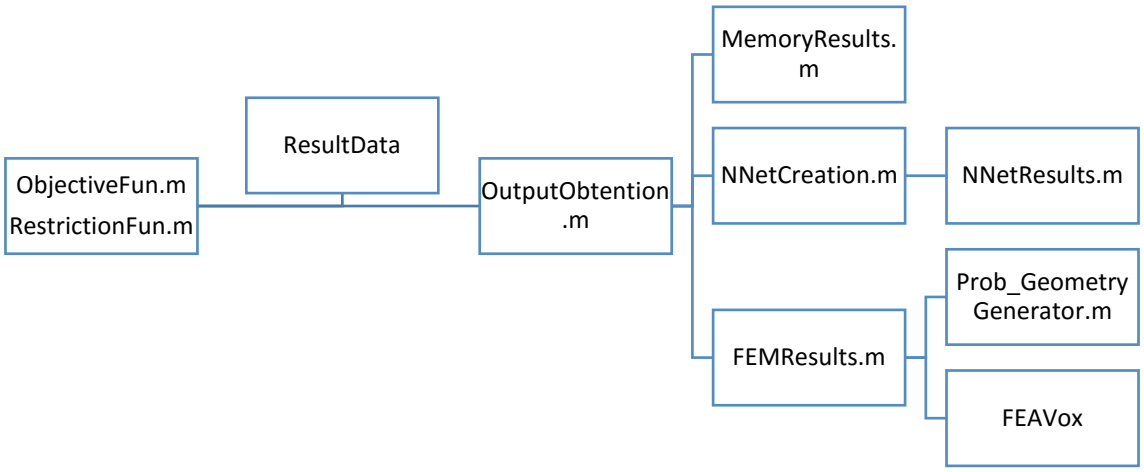

Figura 13. Proceso ObjectiveFun.m y RestrictionFun.m

3.2.3 Prob_GeometryGenerator.m

Como se ha podido observar en los apartados anteriores no se ha detallado nada en cuanto a la generación de las geometrías, esto es porque en el proyecto se le ha querido dar entidad propia para hacer una descripción lo más detallada posible.

Esta función necesita, para funcionar correctamente, tres tipos de datos de entrada, ellos son:

- Las variables de diseño.
- El caso de diseño, que se explicará a continuación.
- El nombre de la geometría generada.

Para crear las geometrías más complejas se ha usado *GiD*, mientras que las geometrías con pocas variables de diseño (una o dos como mucho) pueden generarse directamente con *MatLab*.

La función encargada de generar la geometría por *MatLab* se llama Geo_*CADGenerator*.m y la propia mediante el uso de *GiD* es Geo_*GiDCADGenerator*.m, ambas funciones se valen de los parámetros de diseño como valores de entrada para su uso correcto.

A continuación, se detallará la función que se vale de *GiD* para generar geometrías ya que es la de mayor interés y versatilidad.

Antes de describir el funcionamiento de esta función es necesario explicar los archivos que se han debido crear previamente para facilitar su uso. *GiD* es un software que permite diseñar geometrías mediante su entorno gráfico. Al tiempo que se usa el entorno gráfico, *GiD* graba todos los pasos realizados hasta generar la geometría en un archivo de texto. En este archivo se encuentran todas las acciones que ejecuta el usuario en orden, de modo que el software sería capaz de volver a generar la misma geometría simplemente leyendo ese archivo, a modo de archivo de procesamiento por lotes, o fichero tipo *batch*, sin que el usuario tengo que introducir nada de información. En nuestro caso, nos valemos de estos archivos de texto creados al generar una geometría con valores arbitrarios de los parámetros geométricos que la definen, que serían nuestras variables de diseño. Posteriormente podremos crear nuevas geometrías simplemente modificando el valor de los parámetros en el archivo de texto y haciendo que *GiD* procese el nuevo archivo. En este trabajo se ha creado uno de estos archivos parametrizados para cada una de las geometrías consideradas en los análisis numéricos. En estos archivos de texto, los valores numéricos de las variables que se usarán como parámetros son sustituidos por una cadena de texto que indica el nombre de la variable de diseño. La creación de estos archivos (que no pueden ser procesados por *GiD* debido a dichas cadenas de texto) es un trabajo que se realiza previamente a la ejecución del optimizador y solo se debe realizar una vez. Así, la creación automática de nuevos archivos de texto que pueda procesar *GiD* consistirá simplemente en buscar la cadena de texto de cada variable y sustituirla por el valor numérico deseado.

Lo primero que realiza la función Geo_*GiDCADGenerator*.m es abrir el archivo *batch* que contiene la geometría parametrizada que se analizará. Posteriormente se crea un archivo que es simplemente un listado de los parámetros de diseño introducidos en la función que servirá para comprobar que el número de parámetros es el adecuado y que, por tanto, se podrá generar la geometría.

El siguiente paso, es el de, como se indicó anteriormente, cambiar los parámetros incógnita que aparecen en el fichero de texto *batch* por los parámetros numéricos que se han introducido en la función. Una vez hecho esto, y sin cerrar este archivo, se van a añadir unas líneas de código

con el lenguaje propio de *GiD* para que al generar la geometría la exporte como un archivo IGES, que es el formato estándar de intercambio de archivos CAD que se usará.

Con el archivo *batch* ya completo, se procesará con *GiD*. Esto se hace sin ejecutar el entorno gráfico de *GiD*, a fin de reducir coste computacional. Este proceso genera el archivo IGES con la geometría del componente a analizar. Antes de continuar con el tratamiento del archivo, se realiza una comprobación sobre el número de superficies que forman la geometría para ver si no ha habido ningún error que pudiera haber generado alguna geometría defectuosa. Para generar la geometría final, nos valemos de un par de funciones auxiliares que trasformarán ese archivo en formato IGES a una geometría con el formato de *MatLab* y con la que *FEAVox* ya podrá ejecutar su análisis.

El proceso aparece representado en el diagrama siguiente (*ver Figura 14*):

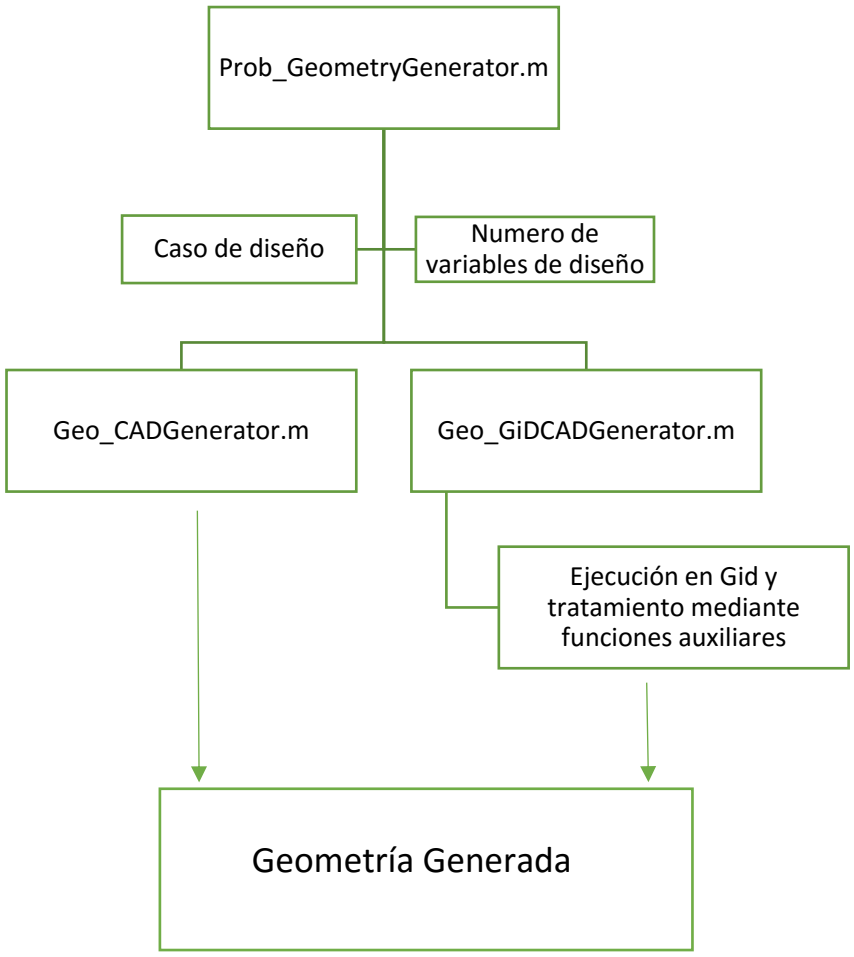

Figura 14. Proceso para la generación de geometrías.

4 RESULTADOS

Este apartado sería la parte central del proyecto en el que se verán los resultados que proporciona el optimizador frente a diferentes situaciones y la comparación entre éstas servirán para comprobar si las mejoras son provechosas para la aplicación o no. Las mejoras que se han hecho frente al proceso del simple cálculo de cada uno de los individuos son, por un lado, el hecho de poder recuperar datos almacenados en la misma ejecución o si se desea en una ejecución anterior y, por otro lado, el uso de redes neuronales artificiales. Independientemente de estas mejoras también se verá la influencia del tamaño de malla sobre los resultados.

Se considerará como solver a cada una de las etapas por las que pasa el optimizador. Podemos considerar 2 tipos de solvers. En el primer solver se ejecuta el algoritmo *fmincon* desde el punto de partida sugerido por el usuario, mientras que en el segundo solver, *GlobalSearch* actúa e inicia el algoritmo *fmincon* tantas veces como puntos se hayan indicado. En los ejemplos se va a considerar que el proceso ha sido exitoso si uno o los dos solver convergen a un valor óptimo.

La primera mejora, que hace referencia a la capacidad de poder recuperar la información de análisis previos, en cuanto a recuperar los datos previamente almacenados es la consecuencia directa de observar el programa en funcionamiento libremente. Lo que se ha observado es que el optimizador llamaba repetidas veces a un mismo individuo, por lo tanto, con realizar los cálculos referentes a ese individuo una vez debería ser suficiente. Sin realizar ninguna ejecución ya se intuye que esta mejora va a suponer un ahorro drástico del tiempo de ejecución al obtener la información de un individuo de manera instantánea dado que había sido analizado con anterioridad, frente a tener que realizar el análisis de elementos finitos de nuevo de una geometría que ya había sido analizada.

Con la implementación del uso de redes neuronales (NNet), el tiempo de ejecución y la precisión de los resultados dependen en gran medida de varios factores adicionales:

- El número mínimo de resultados de análisis provenientes del cálculo de FEM e introducidos como medio para entrenar las redes neuronales, es un factor fundamental. Cuantos más resultados se usen para alimentar la red mayor será posibilidad de disminuir el error de la aproximación realizada frente al cálculo estándar realizado con el MEF. El aumento del número de resultados supone invertir un mayor tiempo en su entrenamiento, pero este tiempo extra es, en general, despreciable frente al coste de un análisis MEF.
- Se ha implementado un bucle que crea y entrena varias NNets para cada variable, eligiéndose como válida la que proporciones el menor error de validación. El número de redes creadas antes de obtener la óptima también supone un factor a considerar, ya que al crear muchas redes se invierte mayor tiempo, que suele quedar compensado por aumentar la probabilidad de obtener una red con un error menor.
- El número de ejecuciones que tiene permitida la red sin ser reevaluada de nuevo. Si se va a crear una red para extraer pocos resultados, tal vez, el tiempo invertido no sea rentable. El rendimiento del proceso de optimización aumentará conforme se obtengan más resultados de análisis a través de redes neuronales y menos del MEF. Por otro lado, el uso excesivo de resultados aproximados proporcionados por la red neuronal puede introducir errores notables, especialmente cuando se usa para evaluar geometrías en exceso diferentes a las usadas en el entrenamiento.

- El número de ejecuciones de FEM que se realizan con una red antes de crear la siguiente, por los mismos motivos que el primer factor.
- Si el optimizador sugiere un individuo que esta fuera del rango de entrenamiento la red dará un error inaceptable, entonces se debería calcular por MEF y, con ese nuevo resultado, volver a entrenar la red.

La forma en la que se ha estudiado la eficacia de cada mejora es mediante la medición de tiempos En concreto se mide el tiempo que tarda en ejecutarse la función *OutputObtention*.m, ya que ésta es la encarga de cualquier cálculo que se deba realizar en el programa. De esta forma se eliminan mediciones de tiempo que no nos ofrecen información útil, como el tiempo durante el que el optimizador organiza los puntos de ejecución o las funciones encargadas de imprimir información por pantalla.

También se usarán otras mediciones para comprobar el rendimiento del programa. Por un lado, se medirá el número de iteraciones que realiza el optimizador. Se ha definido una iteración a cada una de las llamadas a *ObjectiveFun*.m o *RestrictionFun*.m, independientemente que a estas se las ejecute con el mismo individuo como entrada. También se tendrá en cuenta el número de individuos diferentes usados por el optimizador hasta alcanzar el óptimo.

El problema a tratar en este proyecto es el análisis de una sección hueca de superficie interior cilíndrica y exterior desconocida. El problema planteado es pues el de una tubería, sometida a presión constante en su superficie interior cilíndrica, para la que se desea encontrar la forma óptima de su superficie externa. Las dimensiones conocidas de la geometría son: una longitud de 20 unidades y una superficie interna cilíndrica definida por un radio de 5 unidades. Las geometrías se han definido con una superficie exterior aleatoria y ésta será la que vaya modificando sus dimensiones hasta que se alcance el valor óptimo. La selección de estas dimensiones óptimas vendrá dada por el valor de las variables de diseño que minimice la función objetivo y que cumpla con la restricción de tensión máxima de von Mises menor o igual a 2 unidades.

Cada uno de los ejemplos está definido por el número de las variables de diseño, el caso de diseño y por el tamaño de malla en MEF. Un valor de tamaño de malla mayor es equivalente a una malla más refinada con la cual obtendremos valores más precisos y cercanos al analítico. Si el valor del tamaño de malla es menor se usará una malla más basta con la cual se obtendrán resultados con un error considerable. Aumentar en una unidad el nivel de malla implica subdividir cada uno de los elementos (cubos) de la misma en 8 elementos cúbicos iguales. Se han creado varios casos de diseño, los cuales están definidos por el número de variables de diseño y el mecanismo encargado de generar la geometría, tal y a continuación se detallarán:

- Caso 1: Una variable de diseño y generación por MatLab.
- Caso 2: Una variable de diseño y generación por GiD.
- Caso 3: Cuatro variables de diseño y generación por GiD.
- Caso 4: Dos variables de diseño y generación por GiD.

Cada una de las ejecuciones será diferente del resto, pero todas compartirán algunos valores de parámetros que definen la ejecución. Al comienzo de la memoria, se ha explicado que *GlobalSearch* necesita que el usuario decida cuantos puntos de inicio tendrá la ejecución y cuantos se ejecutarán en la misma etapa, para facilitar los análisis ambos valores se han definido con el valor de 10 puntos. Otro de los parámetros que compartirán todas las ejecuciones es el valor de la variable de diseño, éste siempre será 15 independientemente del número de

variables. Es decir, en caso de que la geometría solo se defina por una variable (el valor del radio externo) este valor será un escalar: x0 = 15, mientras que en caso de definirse por cuatro variables este valor será el vector x0 = [15 15 15 15].

Los parámetros que cambiarán en cada uno de los ejemplos serán: el número de variables de diseño, el caso de diseño y el tamaño de la malla. Adicionalmente, cada ejemplo tendrá unas condiciones de ejecución diferentes: sin ningún tipo de mejora calculando todos los individuos, con la posibilidad de buscar la información a partir de datos almacenados y, finalmente, el caso anterior, pero añadiendo los datos de una ejecución anterior y la aplicación de redes neuronales.

Existen pocos problemas de optimización estructural en los que sea posible determinar analíticamente su configuración geométrica óptima, uno de ellos es el problema que usaremos en los análisis numéricos. Se sabe que la solución óptima del problema que vamos a analizar tiene una superficie externa circular. Dado que existe una solución analítica de los desplazamientos y tensiones para tubos sometidos a presión interna constante, es posible determinar el radio externo óptimo, que es igual a 13.68 unidades de longitud, lo que corresponde a un volumen de 2546.93 ud3, que será el valor al que deberían tender los procesos de optimización. Tomaremos esta solución analítica como referencia con la que comparar los resultados numéricos.

4.1 UNA VARIABLE CON TAMAÑO DE MALLA IGUAL A 3

Este sería el ejemplo más sencillo que se podría ejecutar, en el que la geometría simplemente está definida por una variable, siendo esta el radio exterior del cilindro que permanece constante a lo largo de toda la tubería.

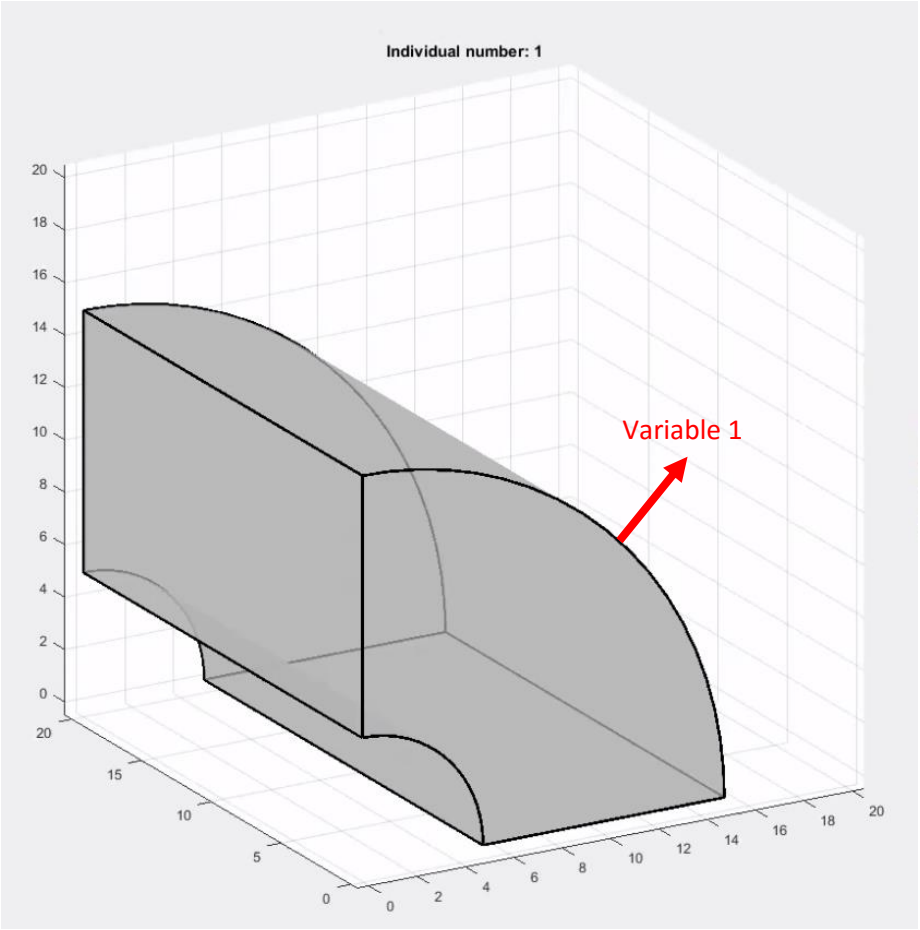

Figura 15. Geometría inicial

Al optimizador se le introduciría esta geometría aleatoria (*ver Figura 15*) con la que trabajar y a partir de esta irá generando individuos hasta que converja en el óptimo.

Recordemos que cada individuo será una configuración geométrica diferente.

4.1.1 Análisis realizados con el MEF

Las condiciones de este ejemplo en particular, sería una ejecución sin ningún tipo de mejora en la que se deba realizar un cálculo MEF por cada una de las iteraciones, aún si aparecen individuos repetidos. En cada uno de los ejemplos nos valdremos de una gráfica en la que se verá la evolución que sigue el optimizador, se representa para cada uno de los individuos el valor del volumen y el de la tensión máxima, como puede verse en la Figura 16.

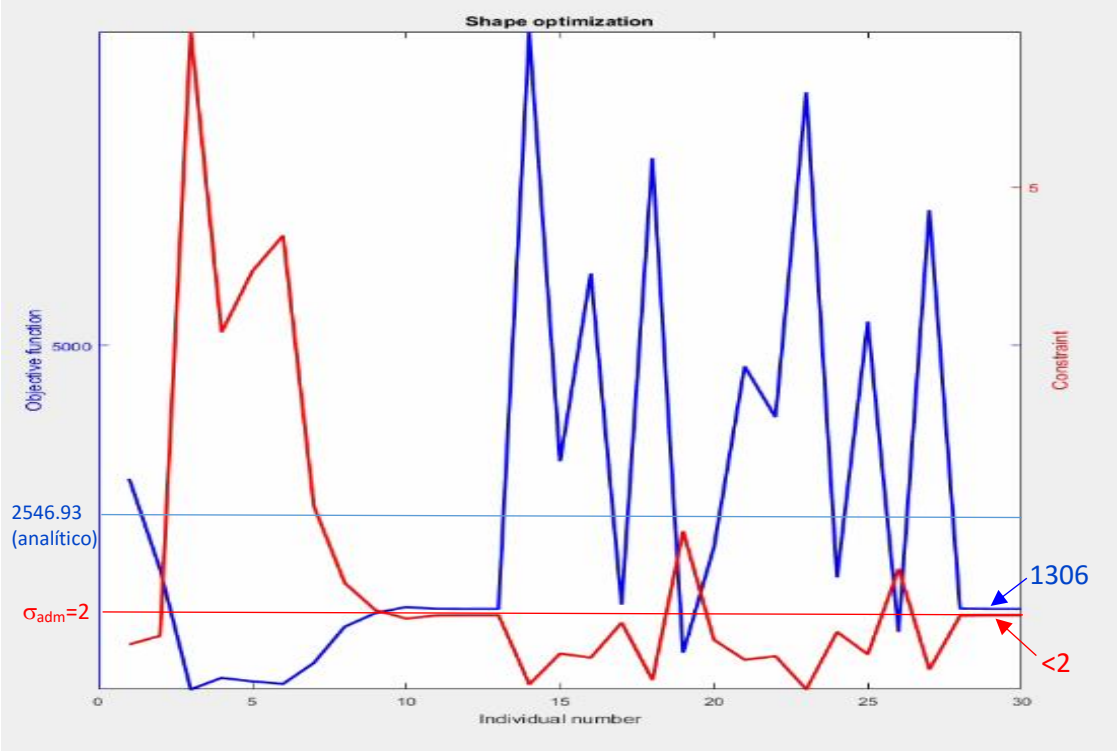

Figura 16. Evolución del proceso.

A partir de este tipo de gráfica se puede conocer cómo ha ido la ejecución, por ejemplo, si ha convergido o no y en caso de hacerlo, la cantidad de veces que lo ha hecho. También se puede conocer la cantidad de individuos que ha probado el optimizador y si ha sido una ejecución con mucha oscilación o ha convergido muy rápidamente.

Las líneas horizontales que aparecen en esta gráfica y en el resto de la memoria son independientes al proceso de optimización. Se han añadido a las gráficas con el fin de evaluar la precisión que tienen sus resultados, ya que estas líneas representan el valor óptimo de la función objetivo (Volumen) y de la restricción (Tensión) analíticas.

De la gráfica que nos encontramos en este apartado en concreto se puede decir que converge en cada uno de los dos solvers ejecutados (Recordemos que el primer solver se refiere al algoritmo *fmincon* lanzado desde la geometría propuesta y el segundo solver es el algoritmo *GlobalSearch* desde su vector de puntos óptimos creados por sí mismo). Observemos que comienza con poca oscilación y en seguida encuentra el valor óptimo y converge, esta etapa es en la que se lanza *fmincon* desde el punto propuesto por el usuario. Tras esto empieza la segunda etapa, se lanza *GlobalSearch* lo cual se traduce en ejecutar *fmincon* desde los puntos que el considere apropiados, de ahí se debe a la extrema oscilación y que la gráfica sea muy picuda, ya que propone punto indistintamente sin seguir ningún orden concreto. Estas dos etapas se repetirán en todas las gráficas que veamos a continuación, con la salvedad de que alguno de los solvers lanzados puede no converger.

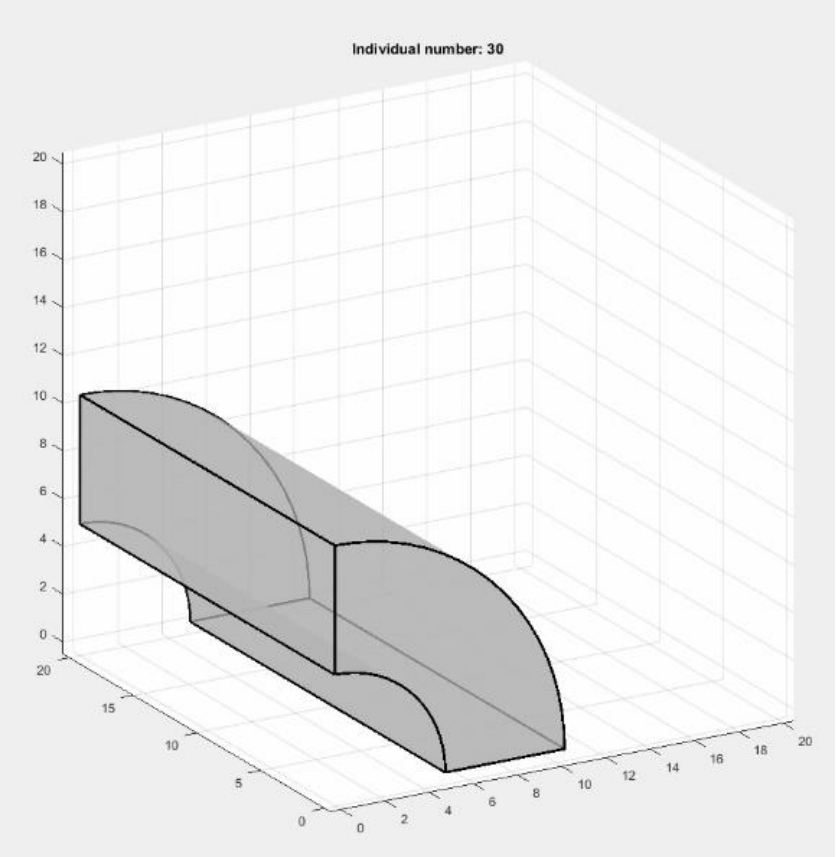

Figura 17. Geometría optimizada.

En la Figura 17 se muestra la geometría óptima de esta ejecución, esta geometría está definida por una sola variable que representa al radio exterior y que tiene como valor 10.4006 unidades. Este valor está alejado del valor analítico óptimo de 13.68 unidades. Esta considerable diferencia es debida al tamaño de la malla. La malla usada en este ejemplo es muy basta y con ella no se pueden obtener resultados con demasiada precisión. Los datos de este problema son una variable de diseño con el primer caso de diseño y con un tamaño de malla igual a 3, dando como resultados los siguientes:

- 149 llamadas al cálculo de resultados.
- Tiempo de ejecución igual a 0.1312 horas, en torno a 7.9 minutos de cálculo.
- Convergencia en radio igual a 10.4006 unidades, lejos del óptimo analítico.
- Convergencia de los dos solvers ejecutados.
- 30 individuos probados.

Que la cantidad de llamadas al cálculo de resultados sea considerablemente mayor al número de individuos o configuraciones geométricas estudiadas se debe, entre otras cosas a que el estudio de cada configuración geométrica exige, al menos 2 llamadas al cálculo de resultados, una para evaluar la función objetivo y restricciones y, otra, para el cálculo de sus derivadas. Adicionalmente el algoritmo de optimización realiza llamadas adicionales para cálculo de resultados, aparentemente para configuraciones geométricas ligeramente distintas a la del individuo que está siendo analizado. La información proporcionada por Matlab es poco clara detallando estos aspectos.

4.1.2 Reutilización de resultados previos del proceso de optimización

Visto este ejemplo y con los resultados de la ejecución bien almacenados se va a proceder a realizar un nuevo ejemplo. En este nuevo caso, se va a proceder una ejecución con los mismos parámetros de entrada que en el caso anterior, pero en esta ocasión el optimizador tendrá la capacidad de buscar en los análisis ya realizados individuos repetidos, con el fin de ahorrar cálculos de MEF.

Observando la gráfica (*ver Figura* 17) vemos que la ejecución es prácticamente idéntica y por lo tanto podemos extraer las mismas conclusiones que con el anterior ejemplo, la única diferencia apreciable entre estos dos ejemplos vendrá a la hora de analizar los resultados obtenidos.

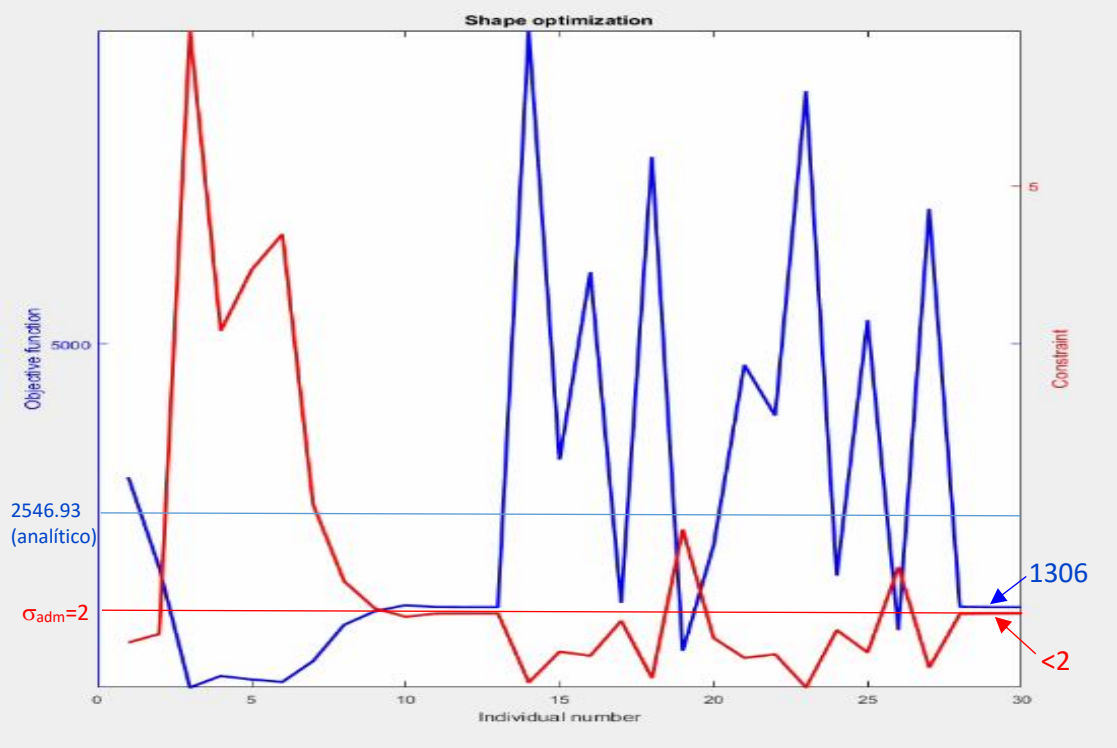

Figura 18. Evolución del proceso.

Los datos de esta ejecución son una geometría con una variable de diseño, el primer caso de diseño y un tamaño de malla igual a 3, en este caso, se le ha permitido al optimizador que busque entre la información ya obtenida durante ese proceso de optimización. Estos parámetros de entrada dan como salida los siguientes resultados:

- 149 llamadas al cálculo de resultados.
- Tiempo de ejecución igual a 0.0252 horas, en torno a 1.5 minutos de cálculos.
- Convergencia en radio igual a 10.4006 unidades, lejos del óptimo analítico.
- Convergencia de los dos solvers ejecutados.
- 30 individuos probados.

Lo que llama la atención es la diferencia en el tiempo de ejecución de los dos ejemplos, siendo el tiempo de este nuevo análisis un 80% menor que al anterior, lo cual significa que esta mejora sí que parece realmente útil.

4.1.3 Reutilización de resultados de un proceso de optimización anterior

El próximo análisis a ejecutar, parte del anterior pero además se le ha añadido la capacidad para buscar datos almacenados en la matriz de resultados de una ejecución anterior. Este análisis converge en el mismo valor óptimo que los anteriores, pero ha sido sorprendente el hecho de que realiza muchas más llamadas a las funciones de cálculo para probar más individuos de los que ya disponía, aun así, el tiempo de ejecución es menor al resto. En la gráfica se puede ver su evolución (*ver Figura 19*) y posteriormente se recolectará la información del análisis.

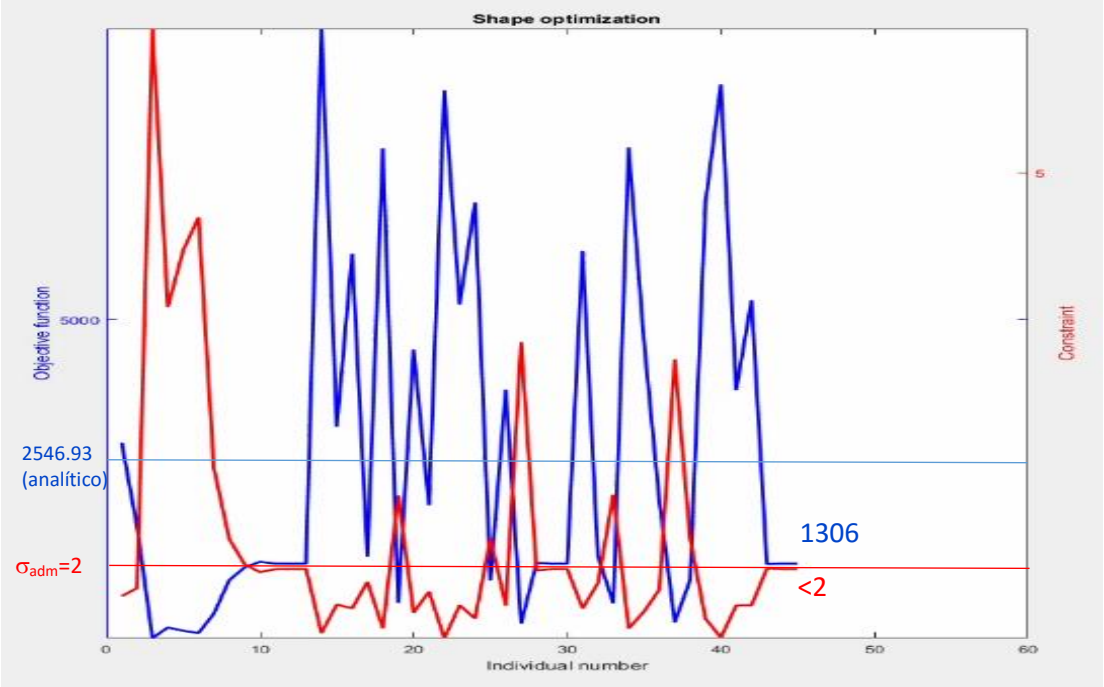

Figura 19. Evolución del proceso.

Analizando la gráfica parece que haya convergido tres veces en lugar de dos como en los casos anteriores, lo cual parece ser a causa de que el optimizador lanza una nueva ejecución a partir del punto de convergencia del segundo solver.

Los datos de entrada de este problema son exactamente los mismos que los anteriores con la salvedad de que se le ha permitido al optimizador buscar la información de un análisis anterior. Dando como resultados los siguientes valores:

- 302 llamadas al cálculo de los resultados.
- Tiempo de ejecución igual a 0.012 horas, menos de un minuto de cálculos.
- Convergencia en radio igual a 10.4006 unidades.
- Convergencia de los dos solvers.
- 45 individuos probados.

Lógicamente esta ejecución tarda menos tiempo en converger ya que dispone de una amplia biblioteca de resultados por los que buscar, lo llamativo de este ejemplo es que llama a nuevos individuos aún habiendo convergido.

4.1.4 Creación de la geometría con GiD

Por último, se ha querido hacer un nuevo ejemplo pero en este caso variando el caso de diseño al segundo, en el cual la geometría se genera mediante GiD, este análisis es exactamente igual que el segundo de los vistos en este apartado, salvo por lo ya comentado.

Observando la gráfica (*ver Figura 20*) vemos que aparecen ciertas diferencias en comparación a los otros. Por un lado, aparecen saltos entre las líneas lo cual se debe a alguna geometría errónea que no se ha podido calcular, debido a que GiD realiza algunas apoximaciones para generar la geometría. Por otro lado, parece ser que solo converge una vez en todo el proceso.

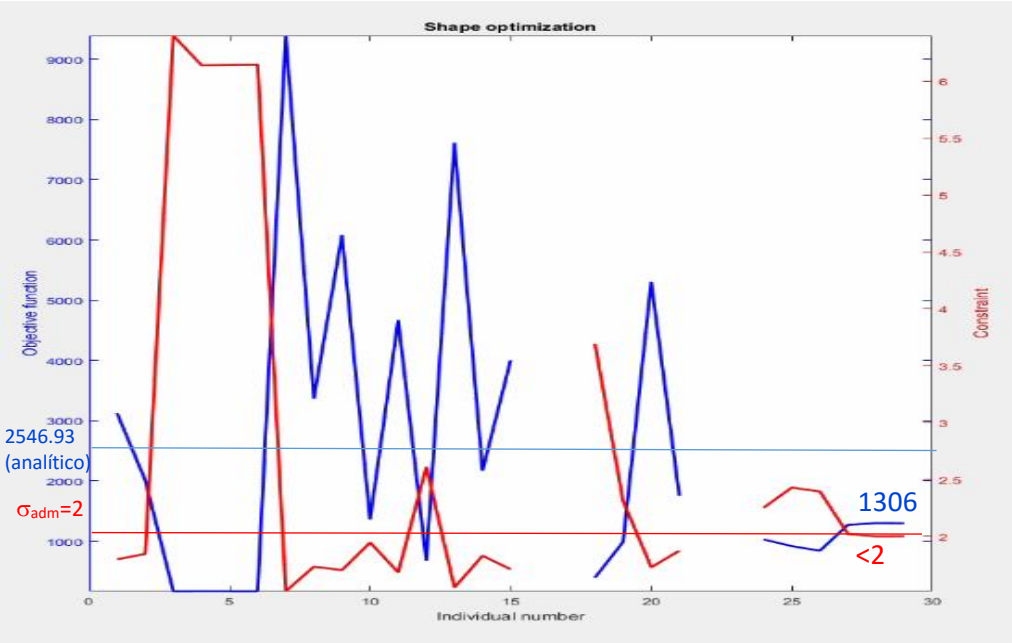

Figura 20. Evolución del proceso.

Los datos de entrada de este problema son una variable de diseño, el segundo caso de diseño y un tamaño de malla igual a tres, este conjunto de datos de entrada genera la siguiente respuesta en los resultados:

- 135 llamadas al cálculo de los resultados.
- Tiempo de ejecución igual a 0.0269 horas, en torno a 1.6 minutos de cálculos.
- Convergencia en radio igual a 10.4037 unidades, lejos de nuevo del óptimo analítico.
- Convergencia de uno de los dos solvers ejecutados.
- 29 individuos probados.

Sólo cabe decir que la geometría óptima (*ver Figura* 21) es prácticamente igual a la de los ejemplos anterior ya que ésta se diferencia en la tercera cifra decimal. El tiempo de ejecución no parece sufrir una variación lo suficientemente grande como para determinar si con uno de los métodos se tarda más que con el otro.

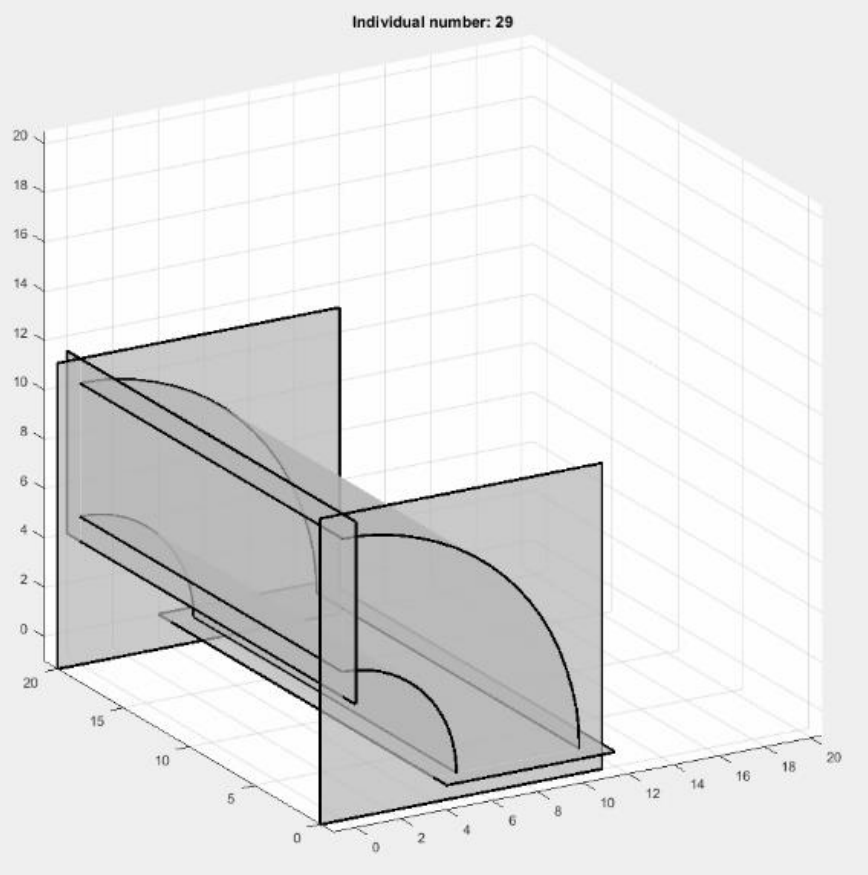

Figura 21. Geometría óptima.

Como resumen de todo el apartado se ofrece la siguiente tabla con los resultados de cada una de las ejecuciones para que sean más fáciles de comparar:

Malla igual a 3	Llamadas a calc.	Tiempo	Xmin	Individuos
Matlab y MEF	149	0,1312	10,4006	30
Matlab y recuperación de datos	149	0,0252	10,4006	30
Matlab, y recuperación de datos más análisis previo	302	0,012	10,4006	45
GiD y recuperación de datos	135	0,0269	10,4037	29

Tras lo visto en este apartado, podemos afirmar que el uso de datos almacenados en memoria puede suponer un ahorro de tiempo drástico y si además se disponen de los datos de una ejecución anterior este ahorro será mayor. También podemos afirmar que el uso de diferentes métodos para generar geometrías no supone una variación en el tiempo de ejecución apreciable, pero sí parece que con las geometrías generadas por *MatLab* aparecen menos errores.

4.2 UNA VARIABLE CON TAMAÑO DE MALLA IGUAL A 4

Los ejemplos que se verán en este apartado simplemente servirán para confirmar la relación existente entre el tamaño de malla y la precisión de los resultados obtenidos.

Con el fin de tener una cantidad mayor de datos se ha ejecutado el mismo problema con los métodos diferentes que dispone el programa para generar la geometría. Ambos problemas podrán recuperar los resultados de análisis previos.

4.2.1 Geometría generada mediante Matlab

Los datos de esta primera ejecución corresponden de nuevo al primer caso de diseño, es decir, el caso donde hay una única variable de diseño (el radio externo), pero se utilizará una malla de nivel 4. Esta variable de diseño ha sido inicializada en el valor de 15 unidades haciendo que la geometría inicial sea exactamente la misma que en el apartado anterior. Al igual que en el apartado anterior se intentarán extraer de la gráfica (*ver Figura* 22) de evolución ciertas conclusiones. Aunque parezca que el proceso converja en una primera etapa al parecer eso no es así ya que se queda en un valor muy próximo al óptimo sin alcanzarlo. Lo mismo ocurre en su segunda etapa, pero este caso se aleja más del óptimo y se puede apreciar con más facilidad.

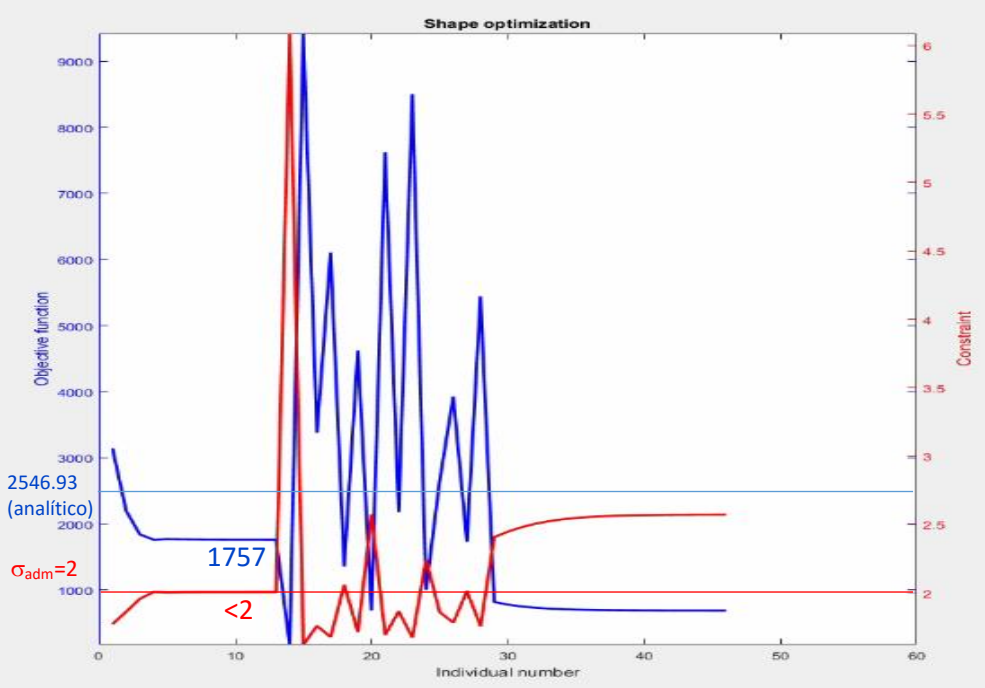

Figura 22. Evolución del proceso de optimización.

La evolución de *fmincon* proporciona para la última geometría de este primer solver un radio exterior de 11.7 unidades, que se aproxima más al valor óptimo (13.68 ud) que, en los análisis anteriores, aunque sigue siendo un valor demasiado bajo. Se puede apreciar, por comparación con los resultados obtenido con mallas de nivel 3, que los resultados mejoran con el refinamiento de la malla proporcionando soluciones más próximo a la analítica.

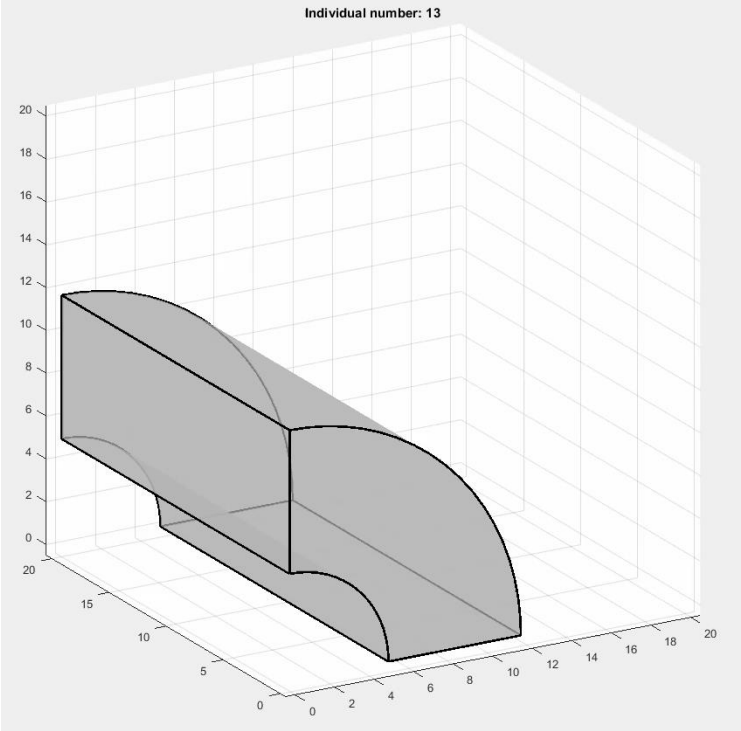

Figura 23.Geometría cercana a la óptima.

Adicionalmente a la geometría que se toma como solución de este proceso, se ha querido añadir también la última geometría analizada por el optimizador con *GlobalSearch* para comprobar la enorme diferencia con el óptimo analítico. Puede verse en la Figura 24, que la geometría, que tiene un radio cercano a 8, difiere mucho de la solución analítica.

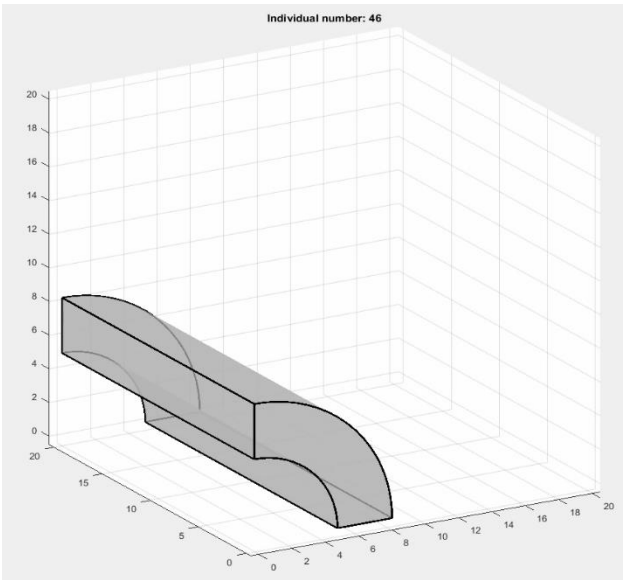

Figura 24. Geometría final, muy alejada de la óptima.

Con los datos de entrada vistos en este ejemplo se generan los siguientes resultados:

- 213 llamadas al cálculo de los resultados.
- Tiempo de ejecución igual a 0.1187 horas, en torno a 7.2 minutos de cálculos.
- No convergencia en radio cercano a 11.7057 unidades, lejos del óptimo analítico.
- No convergencia de ninguno de los dos solvers ejecutados.
- 46 individuos probados.

4.2.2 Geometría generada con GiD

Con el fin de ahondar más en el extraño hecho de la no convergencia del ejemplo anterior se ha querido probar a ejecutar el mismo problema, pero en este caso siendo la geometría dibujada a través de GiD.

Los datos de entrada de este problema son una geometría de una variable de diseño con el segundo caso de diseño y un tamaño de malla igual a 4. Esta serie de parámetros genera una repuesta que se representa con los siguientes resultados:

- 173 iteraciones.
- Tiempo de ejecución igual a 0.1313 horas, en torno a 7.9 minutos de cálculos.
- Convergencia en radio igual a 11.7384 unidades, lejos del óptimo analítico.
- Convergencia de uno de los dos solvers ejecutados.
- 39 individuos probados.

Observando la gráfica de la Figura 25, se puede ver que esta ejecución no dista mucho de la anterior, con la salvedad de que, si logra alcanzar, al menos en uno de sus solvers, la convergencia. El inconveniente que aparece en este ejemplo son una serie de errores en las geometrías que se ven representados por los espacios en blanco de la gráfica. El hecho de que algunas geometrías hayan generado errores, pero que el proceso haya continuado, muestra la robustez de la implementación realizada.

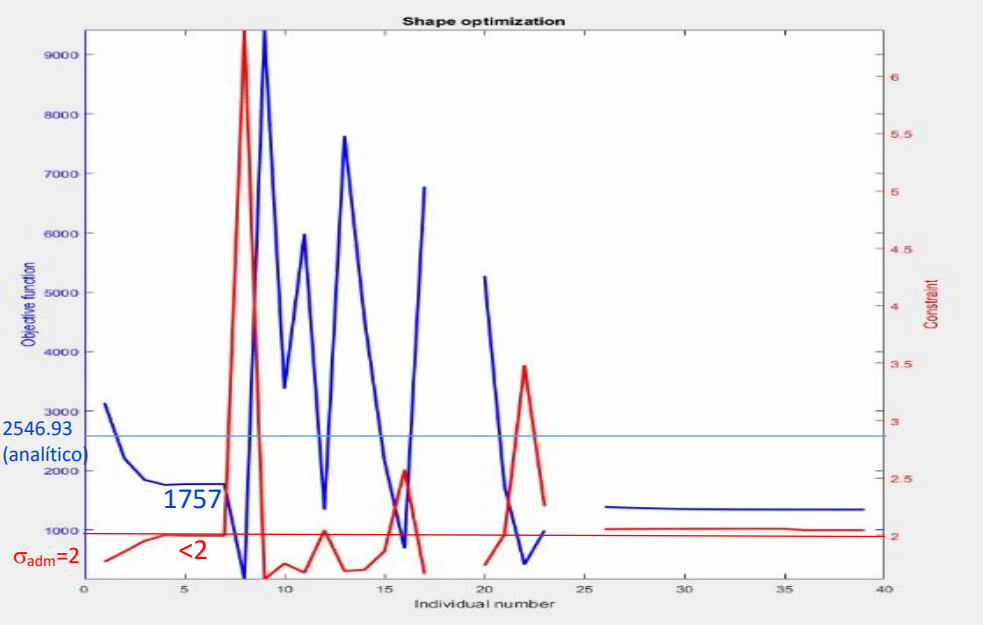

Figura 25. Evolución del proceso de optimización.

La geometría óptima puede verse en la Figura 26, siendo muy similar a la del ejemplo anterior, ratificando que antes se encontraba en un punto muy próximo al óptimo.

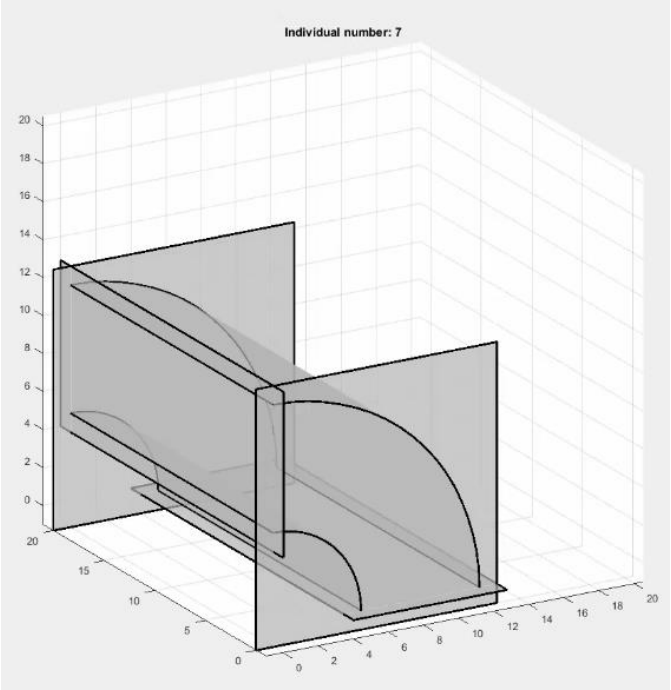

Figura 26. Geometría óptima obtenida.

La única diferencia digna de ser mencionada es que en el ejemplo de la geometría generada por GiD es capaz de converger mientras que en el otro caso no, ya que el resto de resultados son prácticamente iguales.

Con el fin de simplificar la comparación de datos, se adjunta a continuación una tabla resumen de los ejemplos del apartado:

Malla igual a 4	Llamadas a calc.	Tiempo	Xmin	Individuos
Matlab y recuperación de resultados	213	0,1187	±11,7057	45
GiD y recuperación de resultados	173	0,1313	11,7384	39

4.3 UNA VARIABLE CON TAMAÑO DE MALLA IGUAL A 5

Siguiendo con el orden de los ejemplos, ahora es el turno para el análisis de una variable con un tamaño de malla igual a 5. Como es de suponer esta nueva malla nos dará unos resultados más precisos que los vistos hasta ahora, esto se verá con detalle en los resultados. Este nuevo apartado vamos, adicionalmente, a introducir el uso de las redes neuronales como método de reducción del coste computacional. Se ejecutarán análisis con distinto número de datos que alimenten las redes, para ver qué relación existe entre este número y la calidad de la red creada. Aunque nos encontremos en un nuevo apartado la geometría de inicio sigue siendo la misma.

4.3.1 Análisis realizados con el MEF

El primero de los ejemplos a tratar es un problema de una variable con el primer caso de diseño y un tamaño de malla igual a 5, evaluando los análisis mediante el MEF. Observando la gráfica de la Figura 27, vemos que es capaz de converger en su primera etapa con el solver *fmincon*, mientras que en la segunda parece que busca el óptimo en una geometría errónea, ya que no cumple la restricción de tensión. La Figura 28 muestra la geometría óptima proporcionada por *fmincon*.

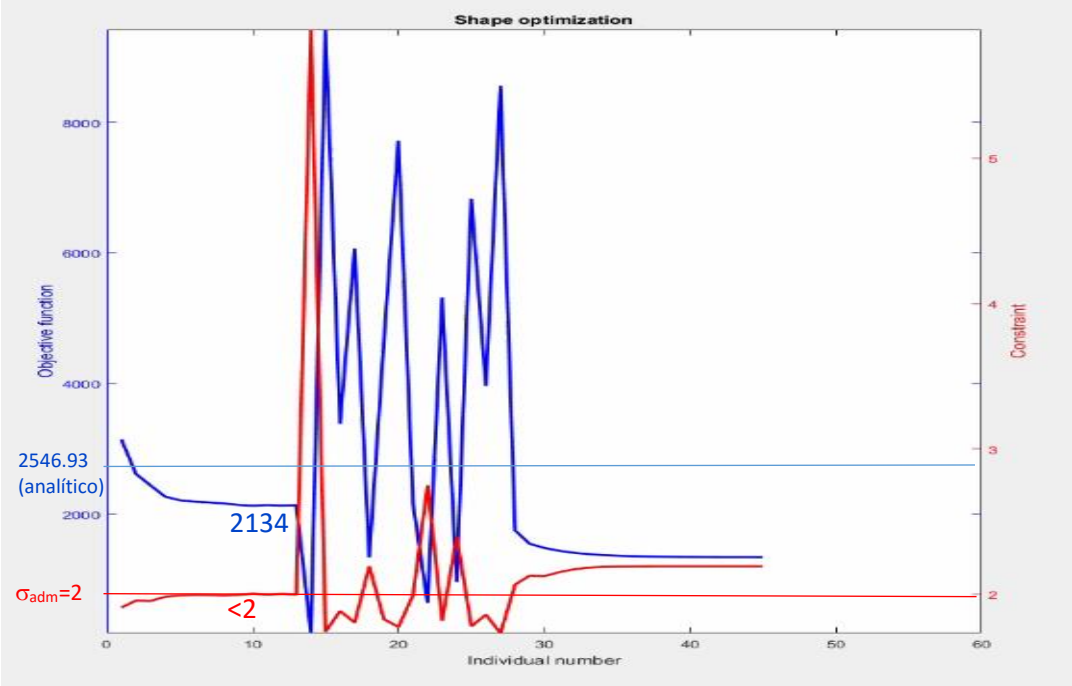

Figura 27. Evolución del proceso.

Sin hacer especial hincapié en la geometría óptima, vemos que en este nuevo ejemplo se acerca mucho más a la óptima analítica, la cual recordemos era un cilindro con radio exterior igual a 13.68.

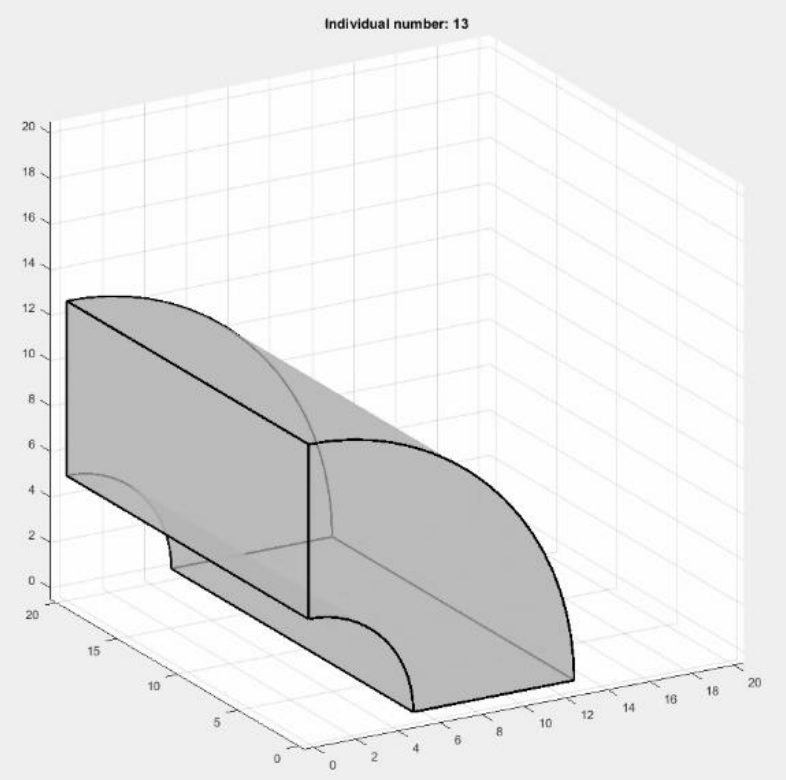

Figura 28. Geometría óptima.

La Figura 29, que aparece a continuación, muestra la geometría a la que tendió *GlobalSearch* al final de su proceso, que difiere notablemente del óptimo analítico.

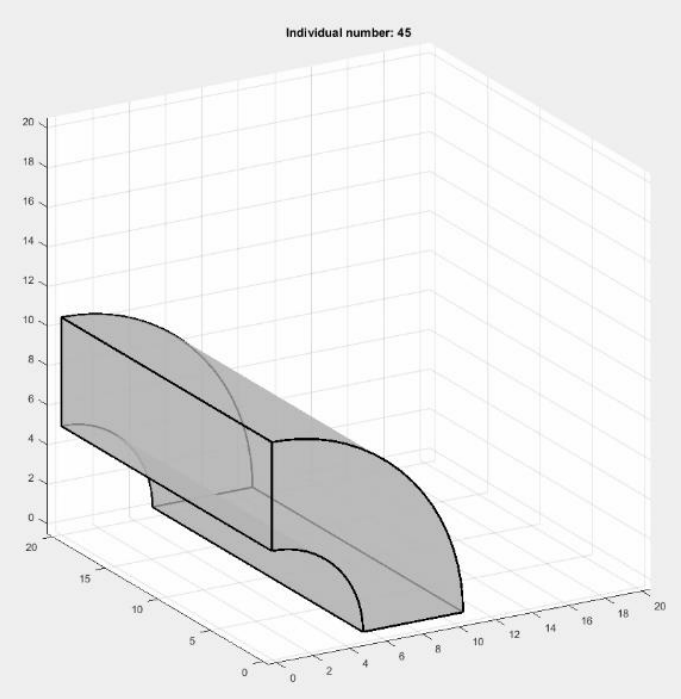

Figura 29. Última geometría de la ejecución.

En este caso el conjunto de datos que definen el problema han generado los siguientes resultados:

- 211 llamadas al cálculo de los resultados.
- Tiempo de ejecución igual a 3.6904 horas.
- Convergencia en radio igual a 12.6839 unidades.
- Convergencia de uno de los dos solvers ejecutados.
- 45 individuos probados.

4.3.2 Reutilización de resultados previos del proceso de optimización y geometría con Matlab

En este análisis se añadirá al último análisis la capacidad de buscar los resultados de geometrías previamente calculadas. Los datos de entrada, por lo tanto, serán los mismos y el algoritmo de optimización seguirá exactamente la misma evolución que en el caso anterior, por lo tanto, no será necesario analizar la gráfica de nuevo. En la Figura 30 también se puede apreciar que la geometría arrojada por el optimizador es idéntica a la que ya se ha visto en el apartado anterior.

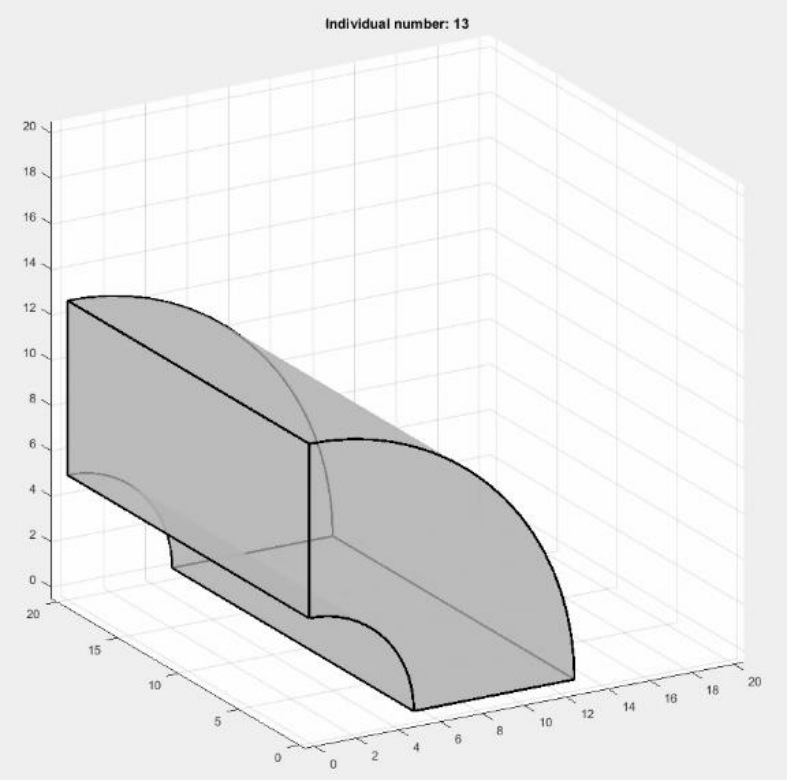

Figura 30. Geometría óptima.

Los resultados que arroja esta nueva ejecución son los siguientes:

- 211 llamadas al cálculo de los resultados.
- Tiempo de ejecución igual a 0.7556 horas.
- Convergencia en radio igual a 12.6839 unidades.
- Convergencia de uno de los dos solvers ejecutados.
- 45 individuos probados.

Analizando los resultados cabe destacar el ahorro que supone la implementación de poder buscar entre los resultados almacenados. A pesar de ser ambas ejecuciones idénticas en el segundo ejemplo se ha ahorrado respecto al primero cerca del 80% del tiempo de ejecución.

4.3.3 Reutilización de resultados previos del proceso de optimización y geometría con GiD

Anexo a este último ejemplo se ha querido realizar la misma ejecución, pero con el caso de diseño en el que la geometría es generada mediante *GiD*. Recapitulando, los datos de entrada definirán un problema de una variable con el segundo caso de diseño y con un tamaño de malla igual a 5, recordemos también que tenía la capacidad de buscar en memoria los análisis realizados Esta combinación de datos de entrada genera una ejecución que puede verse detalla en la Figura 31.

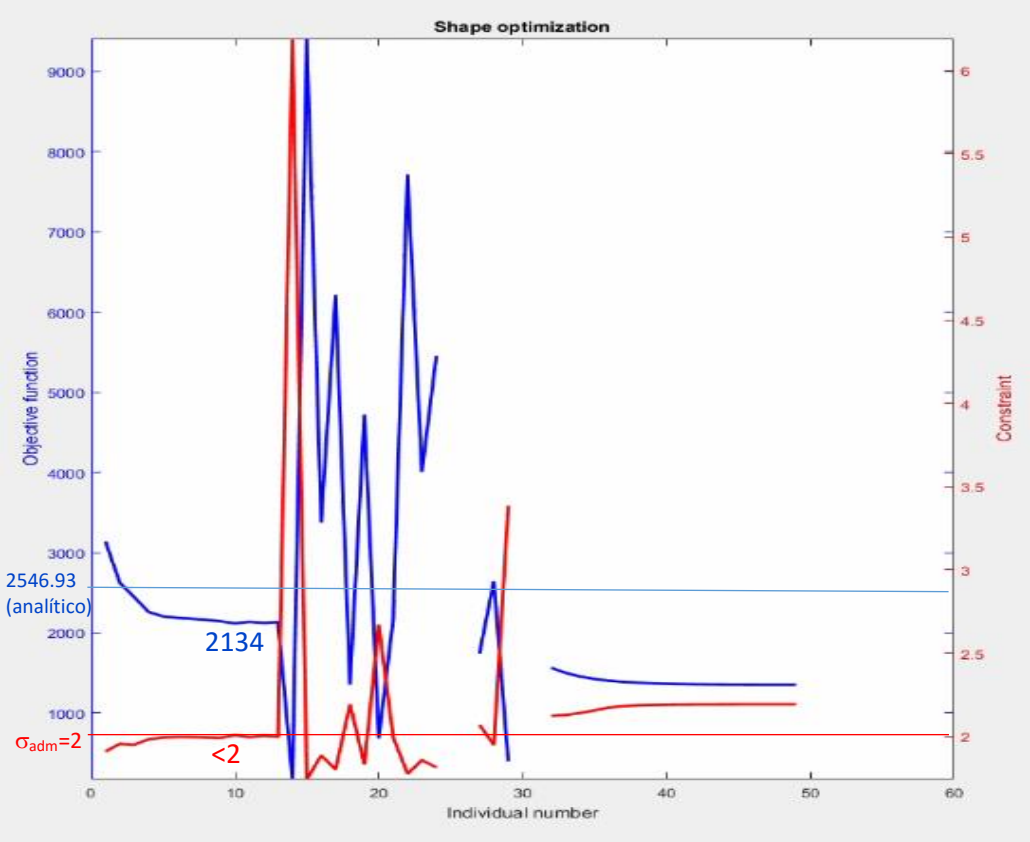

Figura 31. Evolución del proceso.

Cabe destacar que esta ejecución es realmente parecida a la anterior, con la mera diferencia de que en este caso aparecen una serie de errores que se ven representados en la gráfica mediante huecos en blanco. Al igual que antes, es capaz de converger en el primero de los solvers mientras que en el segundo converge hacia una geometría equivocada.

La geometría óptima en este ejemplo puede verse en la Figura 32, siendo virtualmente idéntica al resto de geometrías de este apartado.

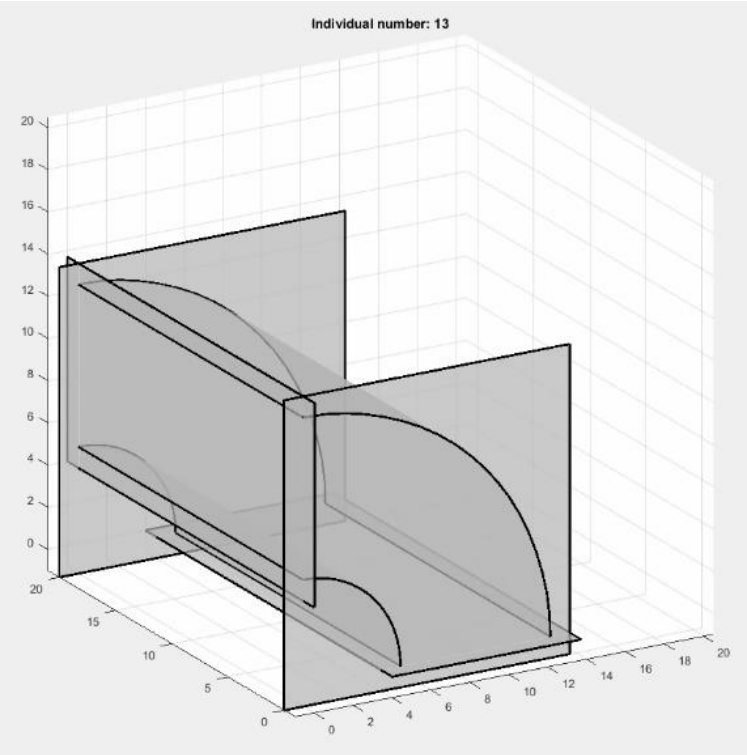

Figura 32. Geometría óptima.

Los resultados de este análisis quedan reflejados a continuación:

- 211 llamadas al cálculo de los resultados.
- Tiempo de ejecución igual a 0.7799 horas.
- Convergencia en radio igual a 12.6919 unidades.
- Convergencia de uno de los dos solvers ejecutados.
- 49 individuos probados.

4.3.4 Redes neuronales artificiales

A continuación, se verán dos ejemplos en los que se ha implementado el uso de redes neuronales. Como se ha comentado, estas redes son capaces de generar resultados al instante a partir de uno o varios valores de entrada. Estas redes neuronales deben ser previamente entrenadas para poder ser utilizadas y para ello serán necesarios un conjunto de datos. Los datos serán las variables de diseño y sus resultados del análisis de MEF. En estos ejemplos veremos la red creada con 30 puntos de entrada y con 45 puntos, con la finalidad de comparar los dos casos.

Comenzando por el primero de los casos, vemos que esta ejecución arroja una geometría óptima exactamente a la de los ejemplos anteriores (*ver Figura 33*).

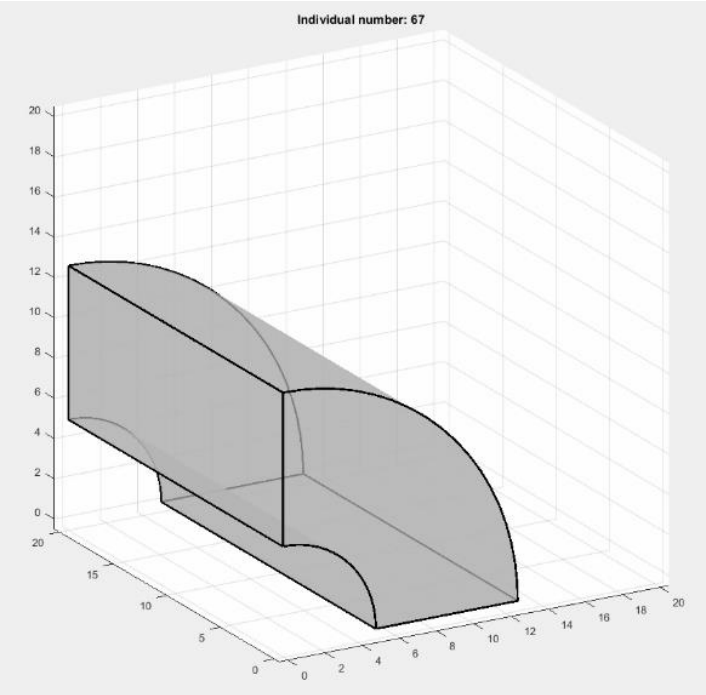

Figura 33. Geometría óptima.

Aunque la geometría óptima sí sea la misma, la gráfica de evolución presenta una serie de diferencias (*ver Figura* 34).

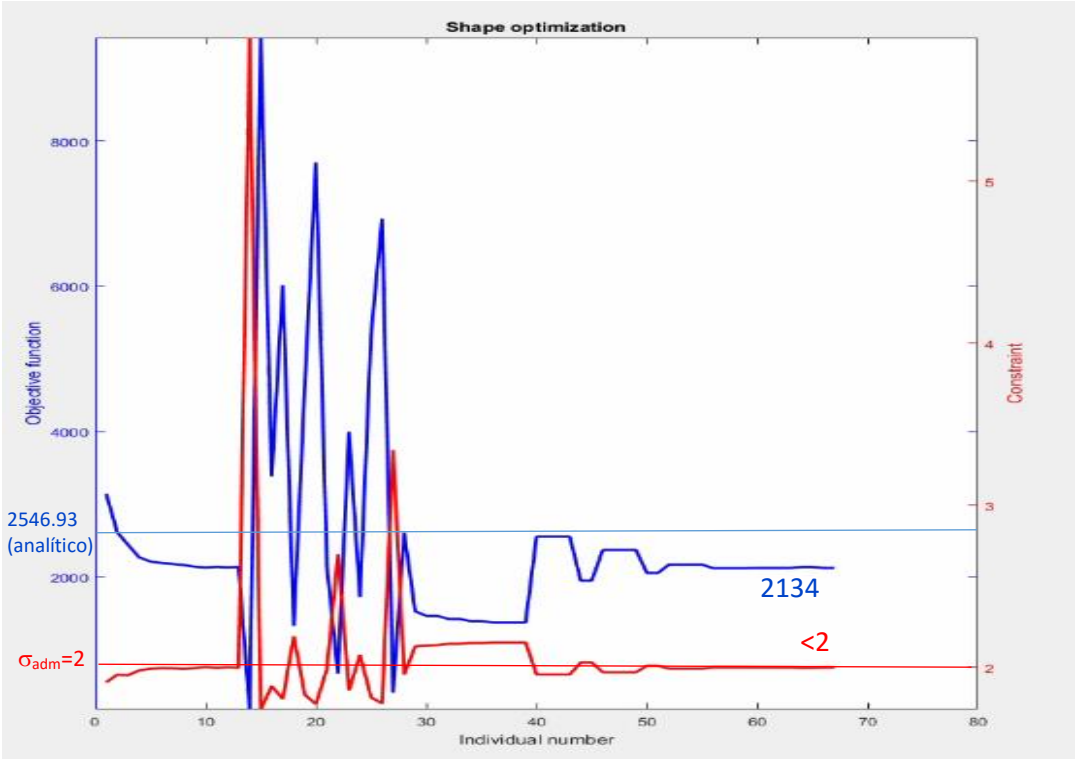

Figura 34. Evolución del proceso.

Leyendo la gráfica de izquierda a derecha vemos que converge una primerza vez y tras esto se lanza la segunda etapa. Después vemos la típica forma picuda, que representa el barrido de puntos en el rango de trabajo. Cuando esta etapa parece que se suaviza entran en funcionamiento las redes neuronales (A partir del individuo 30), y gracias a estas se evita la no convergencia de la segunda etapa, ya que redirige al optimizador por el camino correcto. Gracias a la implementación de redes se ha evitado aquello que ocurria en anteriores análisis y es el hecho de que solo converja uno de los dos solvers lanzados. Analizando en profuncidad la zona en la que actúan las redes, se observa una etapa con una oscilación en torno al punto de geometría óptima que se suaviza conforme nos acercamos al final delproceso.

Recordemos que este ejemplo ha sido realizado con una variable de diseño, el primer caso de diseño y un tamaño de malla igual a 5, adicionalmente se le ha añadido el uso de redes neuronales entrenadas con 30 puntos. Este conjunto de datos genera los siguientes datos:

- 177 llamadas al cálculo de los resultados.
- Tiempo de ejecución igual a 0.5914 horas
- Convergencia en radio igual a 12.6469 unidades.
- Convergencia de los dos solvers ejecutados.
- 67 individuos probados.

Con el fin de que se vea con mayor claridad el funcionamiento de las redes neuronales, se ha tomado una de las redes neuronales creadas y se ha representado en un gráfico. En concreto se ha seleccionado la red que aproxima la restricción, es decir la tensión máxima.

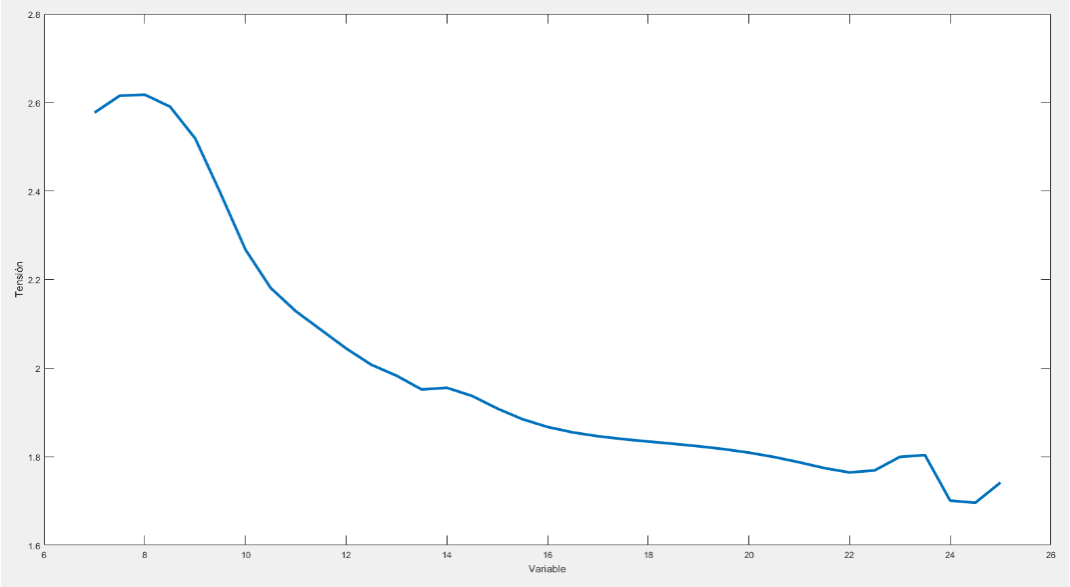

Figura 35. Restricción aproximada mediante redes neuronales.

Como se ve en la Figura 35, las redes neuronales aproximan una función que generá un resultado por cada valor de entrada. En este caso concreto, el valor de entrada correponde al radio externo de la geometría mientras que la salida de la función representa la tensión máxima que aparece en la geometría.

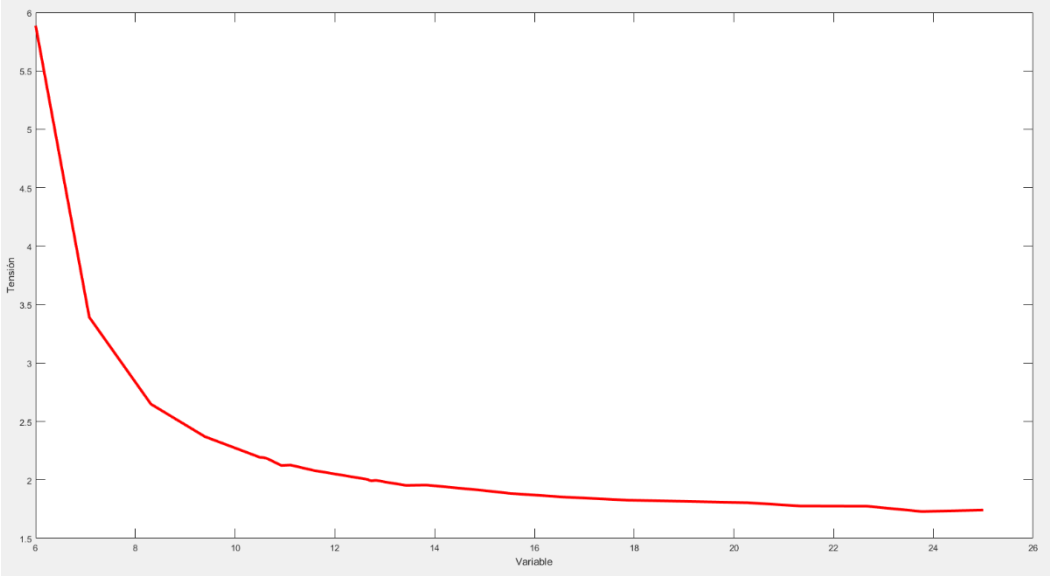

Figura 36. Restricción calculada por MEF.

La Figura 36 representa la restricción calculada mediante MEF. Comparando ambas gráficas vemos que existen claras diferencias. La gráfica que representa la función aproximada parece que difiere enormemente de la otra en sus extremos. Esto es debido a la cantidad de puntos pertenecientes a esa zona, siendo este número mucho menor que los que aparecen en la zona central de la función. Por contra, la aproximación realizada por las redes en los individuos próximos a la geometría óptima no dista excesivamente de la restricción calculada por MEF. Como conclusión, si se quiere obtener los resultados de una geometría próxima a la óptima, las redes no generarán un error excesivo, a medida que esta geometría a calcular se aleje de estos puntos el error aumentará y las redes perderán su utilidad.

La intuición nos dice que, si en un nuevo ejemplo se alimentan las redes con una cantidad de datos mayor, esta aproximación será más precisa que la anterior. En este nuevo caso se lanzará la ejecución con una cantidad de datos de alimentación un 50% mayor que en el anterior ejemplo. Por lo tanto, se necesitarán 45 análisis de MEF antes de poder ejecutar el módulo de redes neuronales. El resto de resultados se darán directamente al final del apartado ya que en este ejemplo lo que se quiere comprobar es la calidad de la red, que se verá representada en la Figura 37. Sin más detalle, la Figura representa una función mucho más próxima a la restricción de MEF, que la del ejemplo anterior.

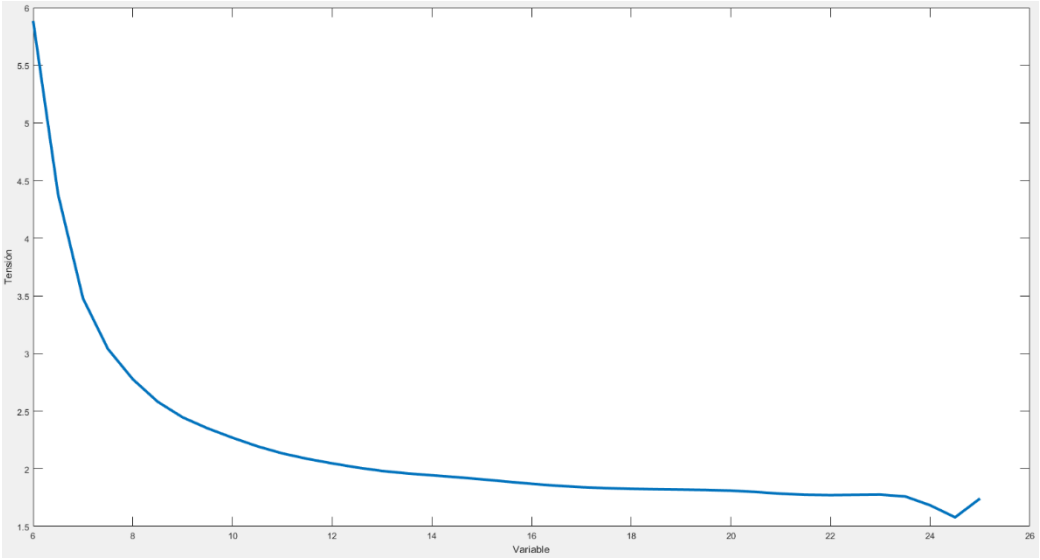

Figura 37. Restricción aproximada mediante redes neuronales.

Para poder comparar fácilmente todos los resultados de este apartado se ha suministrado la siguiente tabla resumen que reúne todos ellos:

Malla igual a 5	Llamadas a calc.	Tiempo	Xmin	Individuos
Matlab	211	3,6904	12,6839	45
Matlab y recuperación datos	211	0,7556	12,6839	45
GiD y recuperación datos	211	0,7799	12,6919	49
Matlab, recup. datos y redes (30 puntos)	177	0,5914	12,6469	67
Matlab, recup. datos y redes (45 puntos)	210	0,7702	12,6748	83

A la hora de extraer conclusiones de la tabla adjunta, ver el ahorro en el tiempo de ejecución que ha sufrido el problema mediante la adición de las diversas mejoras, se ha considerado el hecho de mayor relevancia. Ya se ha comentado que del primer ejemplo al segundo este tiempo se reduce un 80%, pero la implementación de las redes neuronales también supone un mecanismo adecuado para reducir este tiempo de ejecución. Comparando el cuarto ejemplo con el primero de todos, la aplicación de redes y de los datos almacenados supone un ahorro del 85%. En el último ejemplo de todos, en el que la red es más precisa, la aplicación de las redes también supone un ahorro, pero no tan drástico como en el caso de las redes más básicas. A la hora de trabajar con redes neuronales se tiene que tener en cuenta que para conseguir más precisión se necesitan más datos y se invierte un tiempo mayor, la necesidad de precisión o de velocidad de ejecución será decisión del usuario ya que cada problema es único.

4.4 DOS VARIABLES CON TAMAÑO DE MALLA IGUAL A 5

A partir de este apartado, el tamaño de malla siempre va a ser igual a 5, ya que se ha comprobado en los apartados anteriores que el aumento del nivel de malla supone mejorar la precisión de los resultados. Otra de las características que se mantendrá constante a partir de ahora es el método encargado de generar la geometría: ahora solo será posible usar *GiD*. El procedimiento basado en *MatLab* está limitado a una única variable de diseño.

En este apartado la geometría va a estar definida por dos variables de diseño en lugar de una, se verá que esto se traduce en cálculos más lentos y complejos. Las dos variables definirán el arco exterior, encargado de generar la totalidad de la superficie exterior. Una de las variables representa las coordenadas de los puntos situados en el extremo del arco, las otras variables corresponden a las coordenadas de un punto intermedio del arco (*ver Figura 38*).

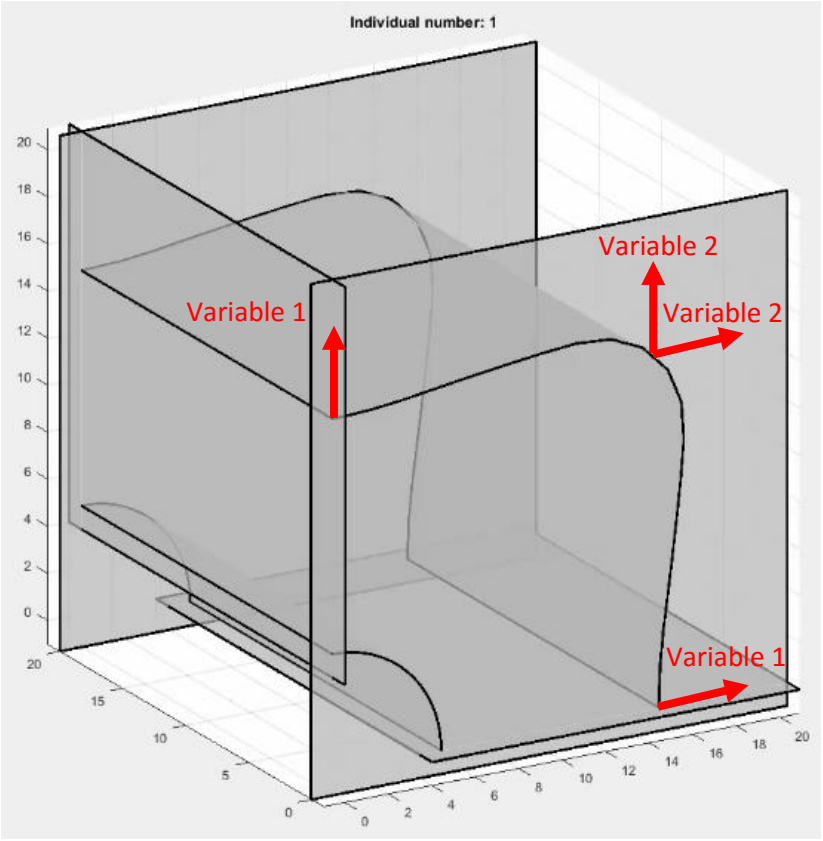

Figura 38. Geometría inicial.

La elección de estas variables ha sido ideada para facilitar al máximo la labor del optimizador, de esta forma, los tras puntos solo pueden desplazarse sobre una recta. El punto situado en el extremo inferior solo puede desplazarse sobre la recta de ecuación y=0. Lo mismo ocurre con el punto situado en el extremo superior, ya que solo puede desplazarse en la recta de ecuación x=0. Al ser las coordenadas del punto intermedio iguales, este solo podrá desplazarse en una recta del tipo y=x.

Una vez definida la geometría inicial de este apartado, se procederá con los ejemplos.

4.4.1 Reutilización de resultados previos del proceso de optimización

La primera ejecución parte de dos variables de diseño, el cuarto caso de diseño y una malla de tamaño igual a 5 como datos de entrada. Adicionalmente a esto se le ha permitido al optimizador buscar en la matriz de resultados los resultados de individuos previos. Recordemos que las variables de diseño han sido inicializadas con el valor de 15, pero en este apartado el escalar pasa a ser un vector con la siguiente forma: x0 = [15 15].

Estos parámetros de entrada arrojan un proceso que tiene la forma representada en la Figura 39.

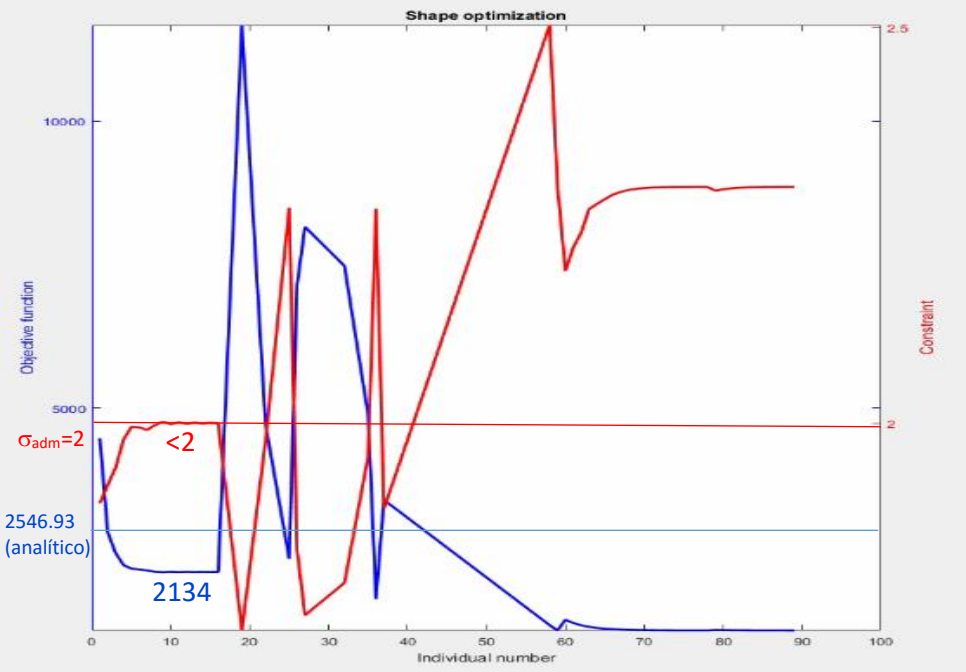

Figura 39. Evolución del proceso.

Al igual que pasaba en otros ejemplos este proceso solo ha sido capaz de converger una vez en su primera etapa, con *fmincon,* mientras que en su segunda etapa se aleja del resultado óptimo. Esta gráfica ha sido modificada para que no se muestren los individuos erróneos, ya que a medida que se aumenta la complejidad del problema aparecen más errores y desvirtúa la gráfica del proceso. Esto puede verse en las líneas con pendientes menores, ya que los resultados intermedios han sido borrados.

A continuación, se muestra la geometría óptima (*ver Figura 40*), la cual es similar a la obtenida en los apartados anteriores. Obsérvese que el optimizador ha proporcionado automáticamente una superficie externa aproximadamente cilíndrica, como corresponde al óptimo analítico.

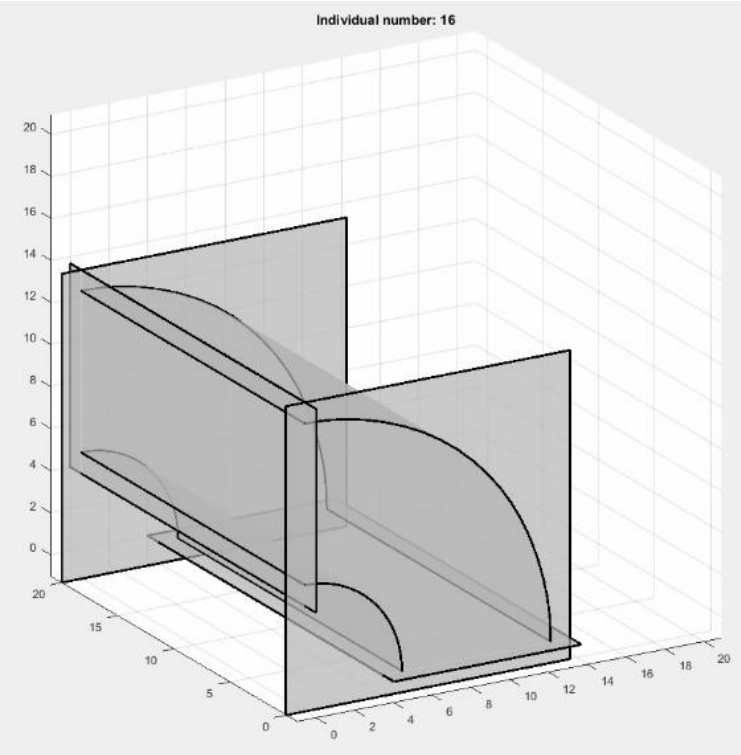

Figura 40. Geometría óptima.

El proceso arroja como resultados los siguientes valores:

- 215 llamadas al cálculo de los resultados.
- Tiempo de ejecución igual a 1.4675 horas
- Convergencia de las variables de diseño en xmin = [12.6802 9.0101] unidades, que correspondería a radios en extremos y punto central de valores [12.6802 12.7422].
- Convergencia de uno de los dos solvers ejecutados.
- 89 individuos probados.

Cabe destacar la diferencia entre el valor obtenido en las variables de diseño y el óptimo, al igual que se ha hecho en apartados anteriores. En el caso de dos variables la geometría óptima analítica estará definida por el siguiente vector, [13.68 9.67]. Sigue apareciendo una diferencia apreciable con respecto al óptimo analítico, aunque aproximadamente igual a la obtenida en los casos anteriores que resultaban más sencillos. Otro dato que resulta necesario mencionar es relativo al del tiempo de ejecución, ya que podemos observar que este análisis tarda bastante más que los de una sola variable.

4.4.2 Redes neuronales artificiales

Los próximos ejemplos partirán de este análisis y se les añadirá el uso de redes neuronales artificiales. Como ya se ha explicado estas redes necesitan datos para ser entrenadas, en el primer ejemplo estas redes se alimentarán con 30 individuos ya que hemos visto que ofrecen resultados aceptables. En el siguiente ejemplo se duplicarán estos datos, entrenando la red con 60 individuos diferentes.

La geometría inicial permanece constante a lo largo de todo el apartado y los datos de entrada con la salvedad de que en este ejemplo se implementarán redes neuronales entrenadas con 30 puntos. Estas condiciones iniciales generan el proceso representado en la Figura 41.

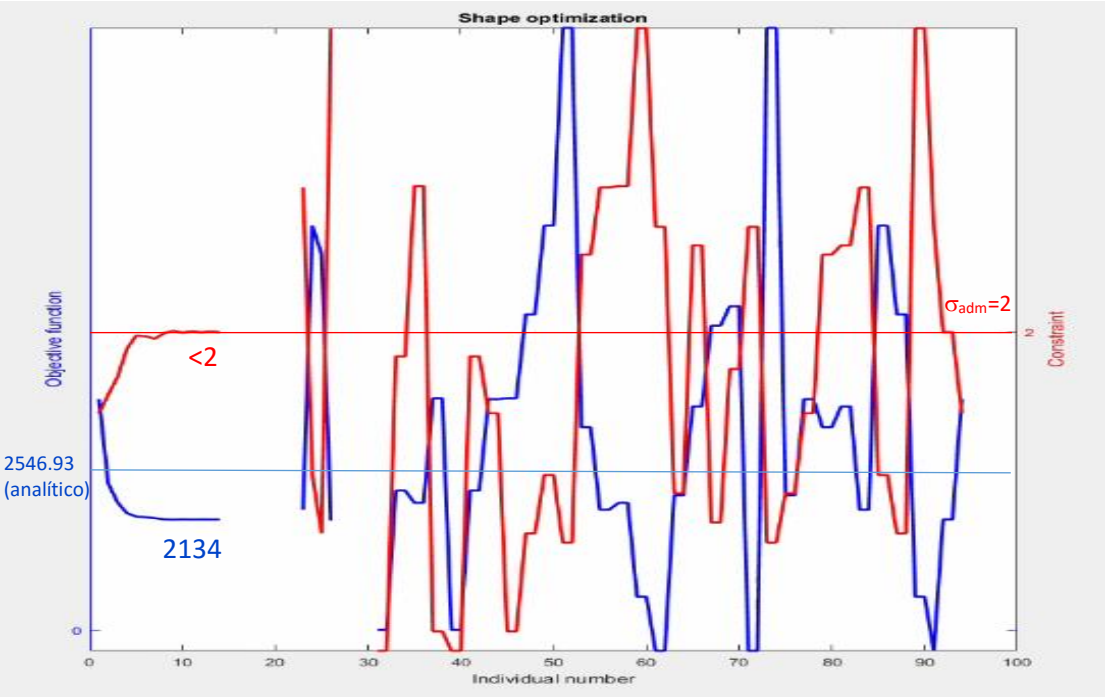

Figura 41. Evolución del proceso.

De la gráfica podemos deducir que la mayor parte del proceso está gobernado por las redes neuronales y que estas no han sido creadas con demasiada calidad ya que genera demasiadas oscilaciones hacia el final del proceso sin llegar a converger en su segunda etapa. Llama la atención la carencia de errores, ya que las redes siempre generan una respuesta, pero esa respuesta puede estar mal aproximada y tener un gran error.

Al igual que se hizo en el anterior apartado, se ha querido representar la red neuronal de la restricción en un gráfico (*ver Figuras 42 y 43, con diferentes perspectivas*). Al haberle añadido en este ejemplo una variable más la representación aumenta en una dimensión, representándose ahora por medio de una superficie. A diferencia de en el apartado anterior, en éste no podremos comparar la aproximación con la función analítica ya que se desconoce cómo se comporta.

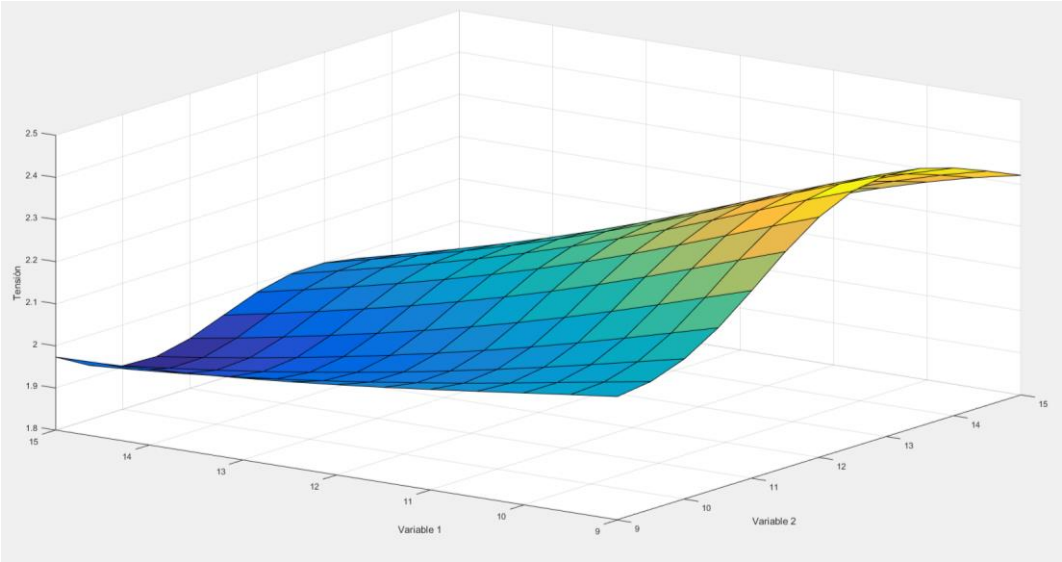

Figura 42. Representación de tensiones proporcionadas por la RNA para dos variables (I).

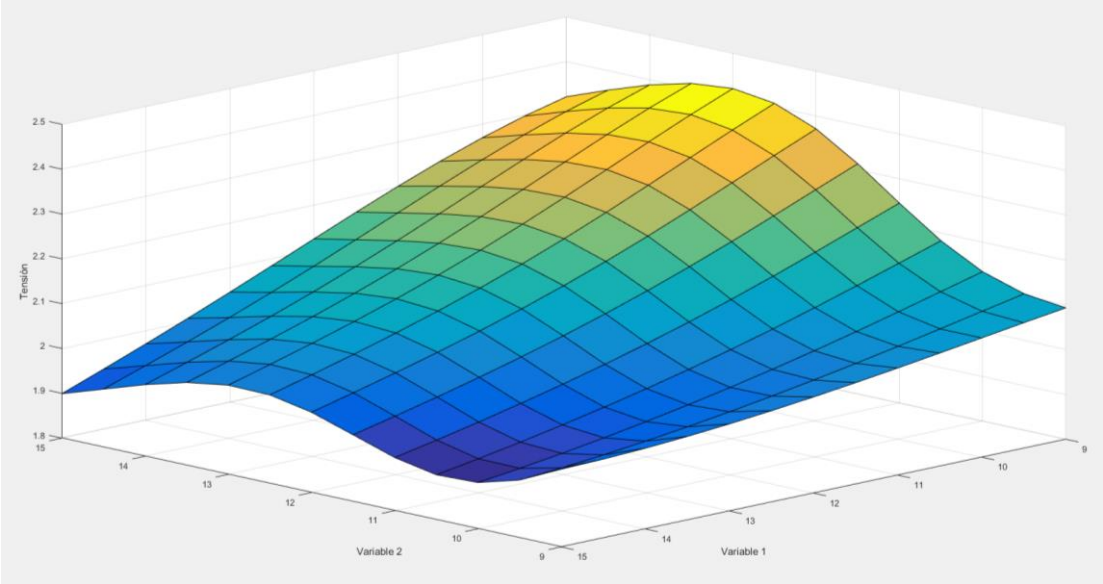

Figura 43. Representación de tensiones proporcionadas por la RNA para dos variables (II).

Con el fin de comprobar la calidad de la red neuronal, se ha pasado un plano horizontal de altura constante que representa la tensión máxima admisible. Si el valor de las variables de diseño que generan la geometría óptima se encuentra en la curva generada por el corte de ambas superficies, diremos que la aproximación, aunque no muy precisa, es adecuada. En la Figura 44 se ha señalado el punto perteneciente a las variables que generan la geometría óptima. Este punto cumple el criterio definido al pertenecer a la curva de intersección.

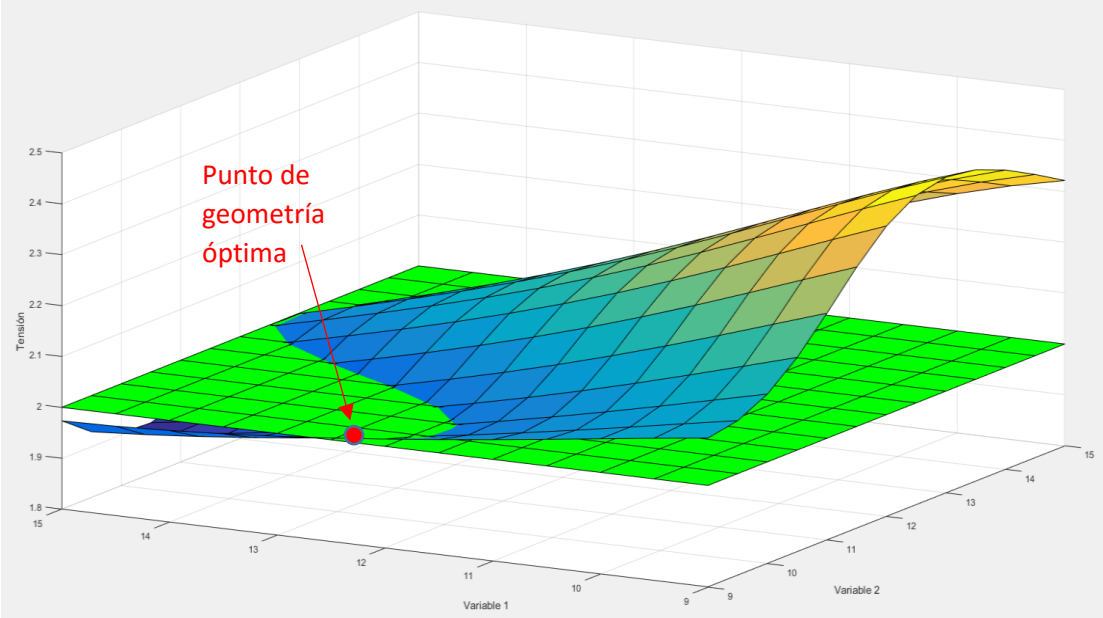

Figura 44. Redes neuronales cortadas por plano de tensión máxima.

Los resultados obtenidos a través de esta ejecución son los siguientes:

- 140 llamadas al cálculo de los resultados.
- Tiempo de ejecución igual a 0.7635 horas
- Convergencia de las variables de diseño en xmin = [12.6805 9.0104] unidades, que correspondería a radios en extremos y punto central de valores [12.6805 12.7426]
- Convergencia de uno de los dos solvers ejecutados.
- 94 individuos probados.

Comparando estos resultados con los del ejemplo anterior se puede afirmar que el uso de redes neuronales suponer un ahorro en el tiempo de ejecución cercano al 50%.

Como ya se ha comentado, se realizará otra ejecución igual que la anterior, pero en este caso con unas redes alimentadas con 60 individuos en lugar de 30. Los resultados de la ejecución se verán al final del ejemplo queriendo hacer hincapié previamente en las redes creadas. Estas redes se ven representadas por las superficies de las Figuras 45 y 46, con perspectivas diferentes.

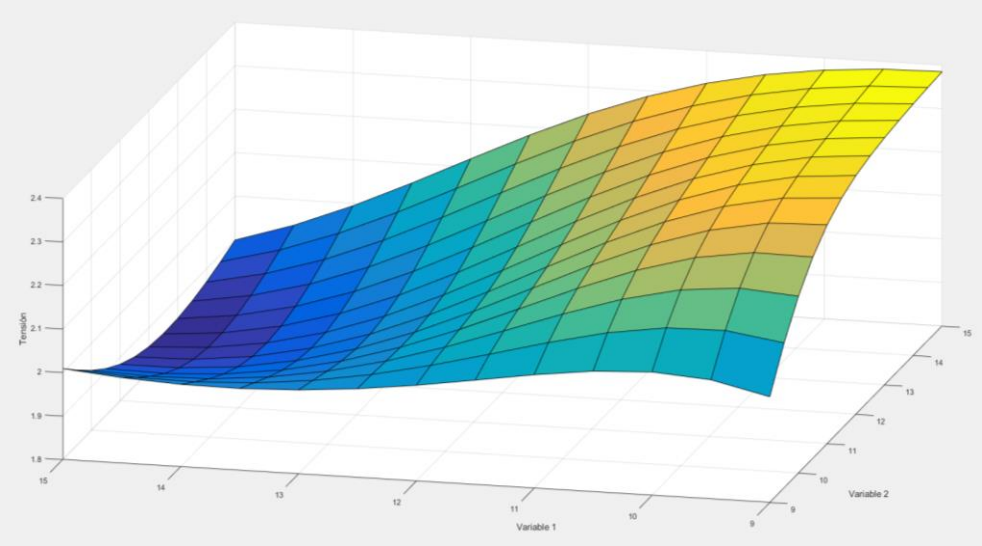

Figura 45. Representación de las redes neuronales (I).

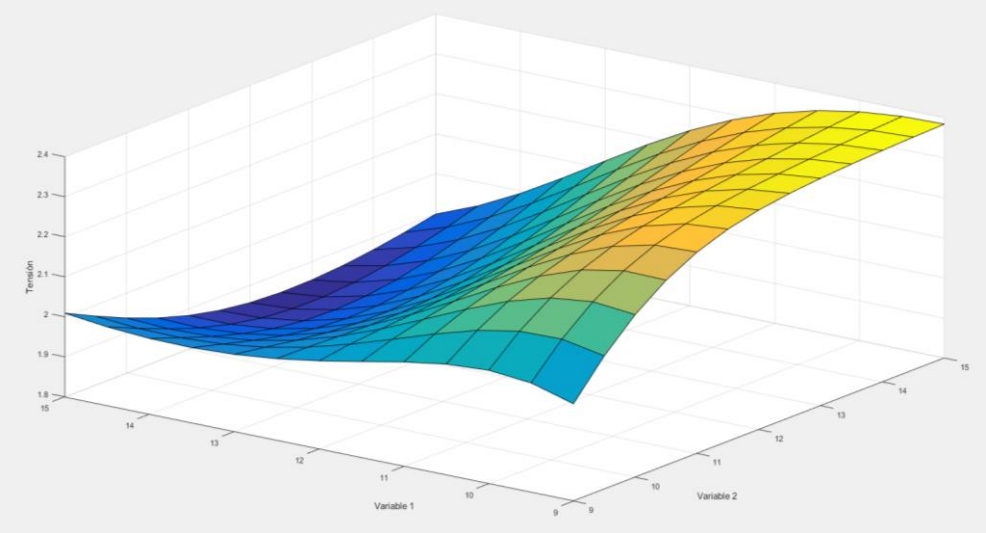

Figura 46. Representación de las redes neuronales (II).

Parece que ahora las redes neuronales ofrecen una respuesta más realista frente a la variación de sus variables al generarse una superficie con curvas más suavizadas, carente de pendientes elevadas. Este hecho es más próximo al comportamiento de un componente estructural real.

Del mismo modo que antes, se ha querido pasar un plano horizontal por la superficie para comprobar que las variables de diseño devueltas cumplen con la restricción. Puede verse el punto perteneciente a la curva perteneciente a ambas superficies en la Figura 47.

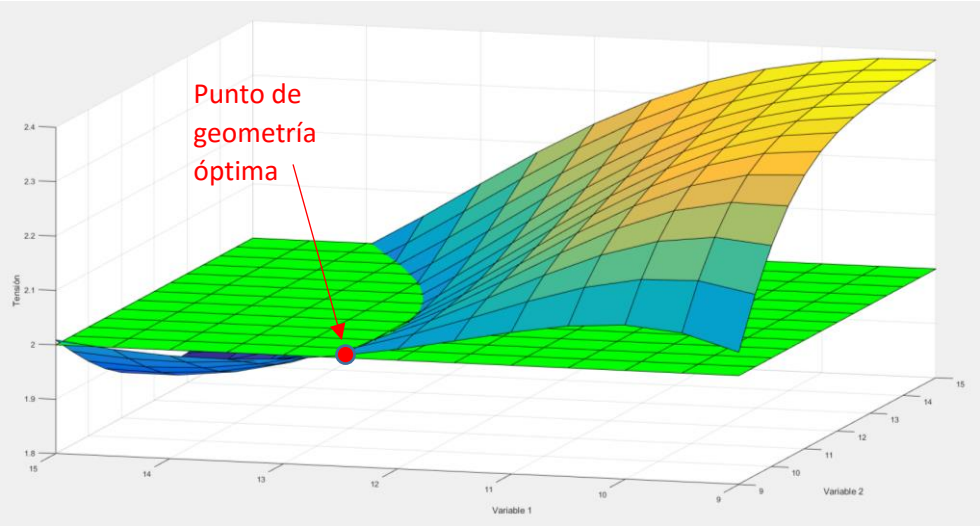

Figura 47. Redes neuronales cortadas por plano de tensión máxima.

Esta nueva ejecución genera una serie de resultados recogidos a continuación:

- 265 llamadas al cálculo de los resultados.
- Tiempo de ejecución igual a 1.2187 horas
- Convergencia de las variables de diseño en xmin = [12.6802 9.0102] unidades, que correspondería a radios en extremos y punto central de valores [12.6802 12.7423]
- Convergencia de uno de los dos solvers ejecutados.
- 156 individuos probados.

En este caso se ha ahorrado tiempo respecto al primer ejemplo, pero no tanto como en el segundo, de ahí a que se deba prestar especial atención a los parámetros que crean y entrenan las redes neuronales.

A continuación, se muestran todos los resultados obtenidos en este apartado en forma de tabla resumen:

Malla igual a 5	Llamadas a calc.	Tiempo	Xmin	Individuos
GiD y recuperación de datos	215	1,4675	[12,6802 9,0101]	89
GiD, recuperación de datos y redes (30 puntos)	140	0,7635	[12,6805 9,0104]	94
GiD, recuperación de datos y redes (60 puntos)	265	1,2187	[12,6802 9,0102]	153

4.5 CUATRO VARIABLES CON TAMAÑO DE MALLA IGUAL A 5

En este nuevo apartado, se estudiará el caso de la geometría definida por 4 variables de diseño. La primera variable corresponde a la componente *x* del extremo inferior, las variables 2 y 3 representan las coordenadas *x* e *y* del punto intermedio y la variable 4 corresponde a la componente *y* del extremo superior (*ver Figura 48*). Estas cuatro variables son suficientes para definir el arco exterior de la geometría y este a su vez generar la superficie exterior.

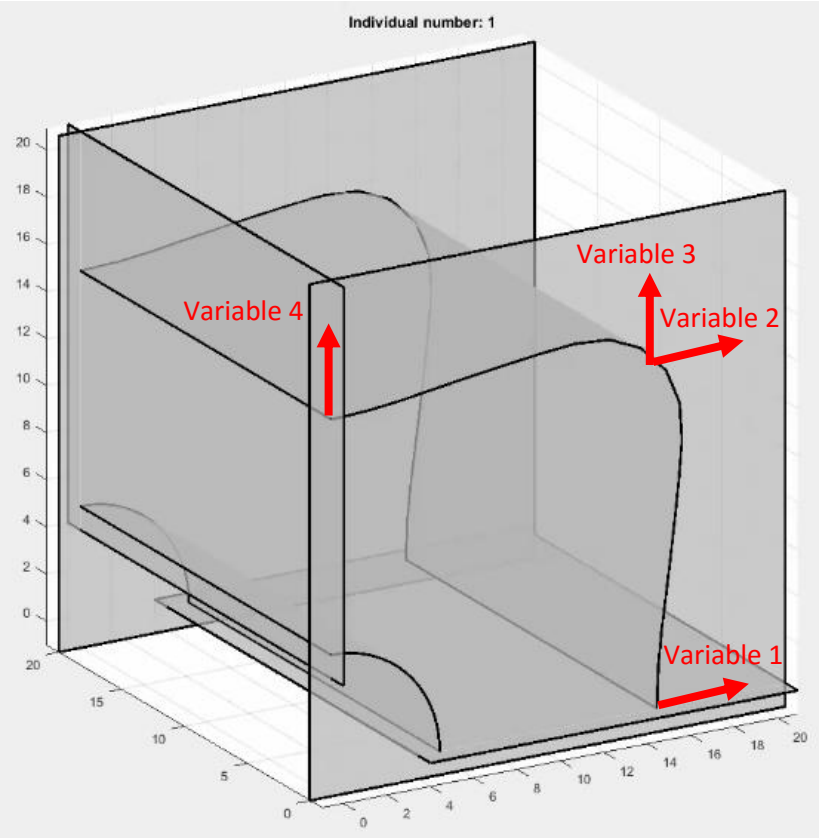

Figura 48. Geometría inicial.

La geometría inicial aparentemente es igual a la del apartado anterior, pero es una simple coincidencia. Al iniciar las variables de diseño en el mismo valor, pueden surgir este tipo de situaciones, pero en realidad esta geometría está definida por un vector fila de tamaño cuatro, cuyo valor es el siguiente, x0 = [15 15 15 15].

4.5.1 Reutilización de resultados previos del proceso de optimización

Con la geometría inicial perfectamente definida, se procederá con la ejecución. El primer ejemplo será una geometría definida por cuatro variables de diseño, el tercer caso de diseño y un tamaño de malla igual a 5.

Esta configuración de datos de entrada arroja la siguiente gráfica de evolución (*ver Figura 49*):

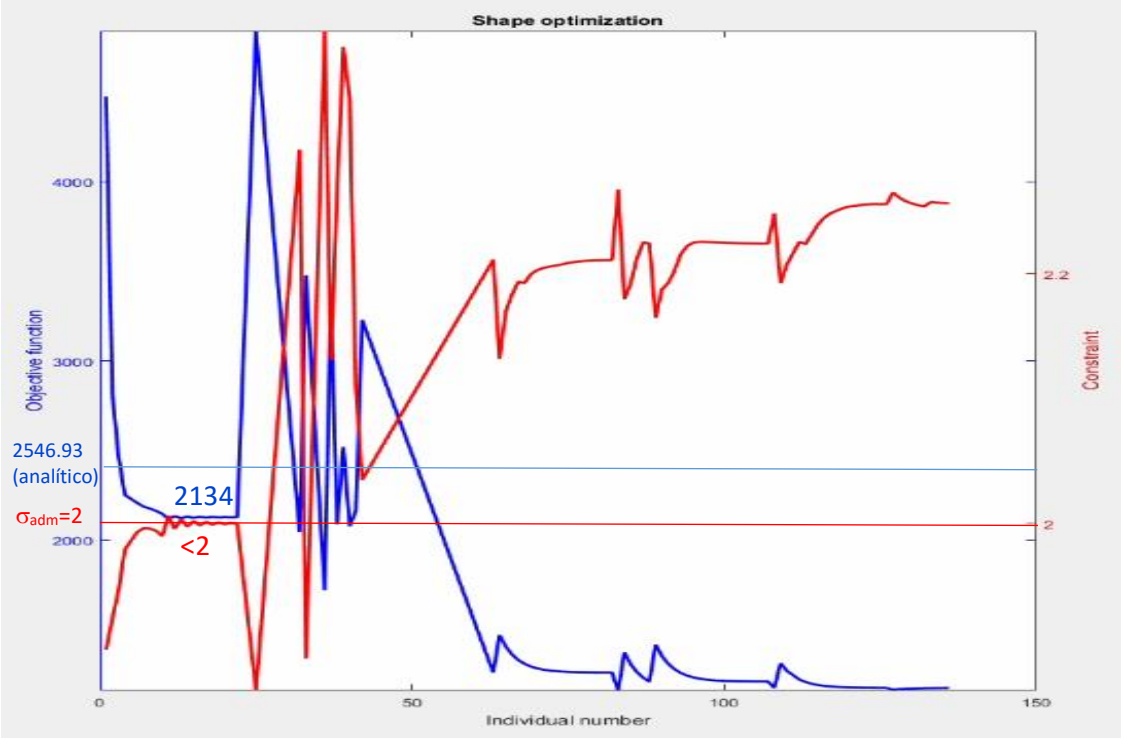

Figura 49. Evolución del proceso.

Como la mayoria de ejecuciones anteriores este proceso solo ha sido capaz de converger en su primera etapa (*fmincon*). Al final de la ejecución el algoritmo continúa el proceso iterativo de la optimización sin encontrar, aparentemente, otro mínimo en el que converger.

Aún no siendo la mejor ejecución posible, el optimizador converge dando como resultado la geometría óptima que puede verse en la Figura 50:

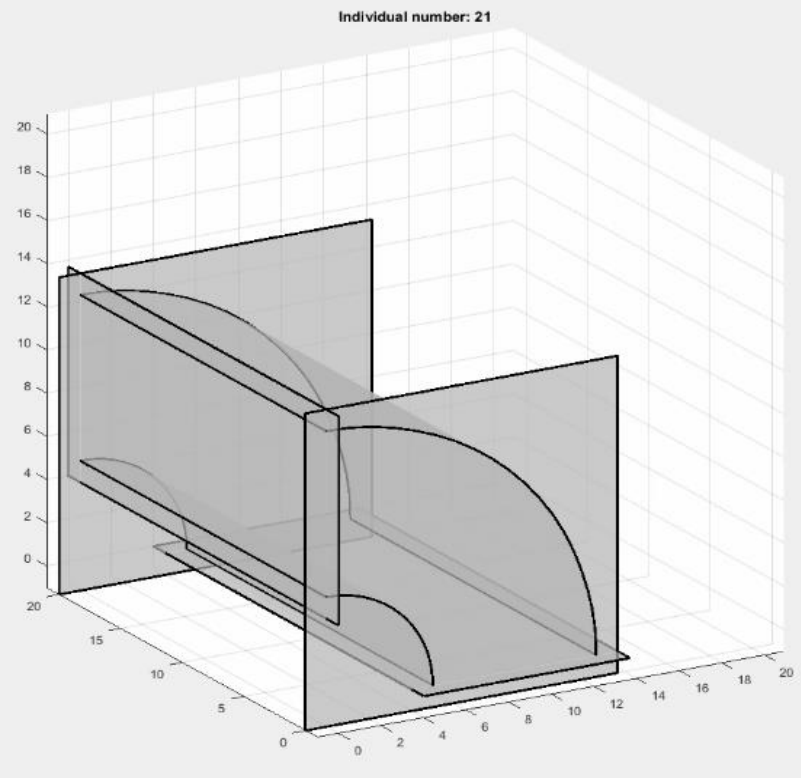

Figura 50. Geometría óptima.

La geometría final parece ser idéntica que en los apartados anteriores lo cual ya se ha indicado que es una buena señal y ratifica la bondad de los resultados obtenidos anteriormente.

Las condiciones iniciales de este problema generan los siguientes resultados:

- 328 llamadas al cálculo de los resultados.
- Tiempo de ejecución igual a 3.9409 horas
- Convergencia de las variables de diseño en xmin = [12.6803 9.0103 8.9103 12.6803] unidades, que correspondería a radios en extremos y punto central de valores [12.6803 12.6720]
- Convergencia de uno de los dos solvers ejecutados.
- 137 individuos probados.

El hecho más destacable es la duración de la ejecución siendo este tiempo mayor que en los ejemplos con menos variables, esto es lógico debido a la complejidad de los cálculos. Comparando el valor de las variables de diseño óptimas de este problema con el de analítico ([13.68 9.67 96.7 13.68]), aparece la misma diferencia que en el resto de apartado que comparten el tamaño de malla. Este hecho solo confirma que con una malla más precisa se obtienes resultados de mayor calidad.

4.5.2 Redes neuronales artificiales

Como es habitual en estos apartados, se añadirá al ejemplo anterior el uso de redes neuronales. En este caso concreto se entrenarán las redes neuronales con 30 individuos diferentes, dando como resultado la gráfica que se ve en la Figura 51.

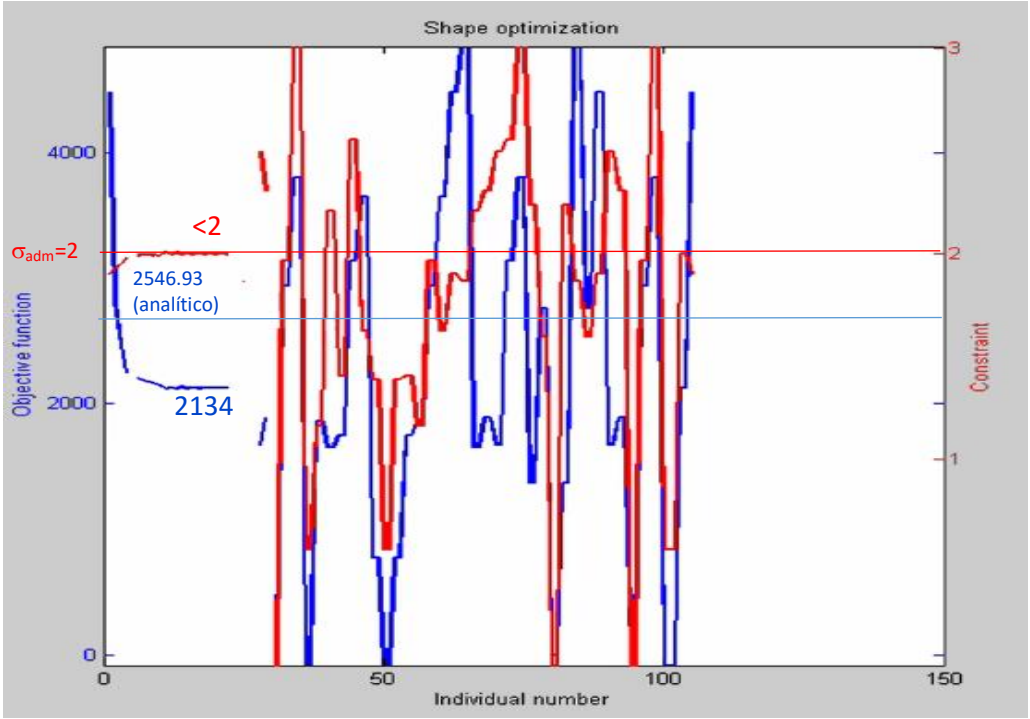

Figura 51. Evolución del proceso.

Al igual que en el ejemplo anterior, este proceso converge una primera vez en su primera etapa, Mientras que en su segunda etapa se produce una oscilación excesiva, lo cual puede ser un síntoma de una red neuronal de poca calidad.

Al ser la geometría óptima exactamente igual al último ejemplo no es necesario volver a recordarla. A continuación, se pretende estudiar la calidad de la red neuronal. Como el número de variables supera el número de dimensiones espaciales es imposible representar la red en un gráfico, así que se decide sustituir un individuo de respuesta conocida en la red. El individuo del que seguro se conoce su valor de restricción es el óptimo ya que estamos condicionando esa respuesta, la cual debe ser igual a 2 (*ver ecuación 5*).

$$bestNNetStr(xmin'^{3}) = 1.9992 \cong 2$$

Ecuación 5. Comprobación de la calidad de una red neuronal.

[3] El apóstrofe denota la traspuesta de una matriz, en este caso la red solo admite valores de entrada con forma de columna, siendo el vector *xmin* suministrado por el programa un vector fila.

Con una diferencia en la cuarta cifra decimal, podemos afirmar que las redes funcionan bien en valores próximos a la geometría óptima, pero no se puede afirmar lo mismo en individuos alejados de esta zona.

Los resultados obtenidos en este análisis son los siguientes:

- 163 llamadas al cálculo de los resultados.
- Tiempo de ejecución igual a 1.3008 horas
- Convergencia de las variables de diseño en xmin = [12.6811 9.0108 8.9108 12.6811] unidades, que correspondería a radios en extremos y punto central de valores [12.6811 12.6727]
- Convergencia de uno de los dos solvers ejecutados.
- 105 individuos probados.

El uso de redes neuronales supone en este caso un ahorro en el tiempo de ejecución cercano al 35%.

Aunque en los otros apartados se haya hecho, en éste no se intentará obtener una red neuronal más refinada ya que al ser imposible representarla no se podría comprobar la mejora en su calidad.

Tabla resumen del apartado:

Malla igual a 5	Llamadas a calc.	Tiempo	Xmin	Individuos
GiD y recuperación de datos	328	3,9409	[12,6803 9,0103 8,9103 12,6803]	137
GiD, recuperación de datos y redes (30 puntos)	163	1,3008	[12,6811 9,0108 8,9108 12,6811]	105

El único hecho destacable en este apartado es el ahorro producido por la implementación de redes neuronales, haciendo el análisis un 66% más rápido.

5 CONCLUSIONES

Se ha conseguido implementar satisfactoriamente el optimizador en el software del departamento, siendo este el objetivo principal de este trabajo. Una vez éste ha funcionado correctamente (con ciertas reservas) el objetivo ha pasado a ser el de reducir considerablemente el tiempo de ejecución. Para ello se le ha dotado de ciertas mejoras. Estas mejoras son concretamente la capacidad para buscar y recuperar resultados de análisis previos y la implementación de redes neuronales artificiales.

La capacidad para reutilizar resultados ya calculados en la propia ejecución ha supuesto una gran fuente de ahorro y si a estos datos almacenados le añadimos los de una ejecución anterior, el ahorro es aún más notable.

Por otro lado, mediante el uso de redes neuronales artificiales se ha probado la versatilidad y capacidad de estas, ya que representan una forma de obtener nuevos resultados de una manera inmediata. A consecuencia de su rendimiento, se ha obtenido un ahorro de tiempo significativo respecto a ejemplos sin redes neuronales.

Estas mejoras han supuesto un aumento de eficiencia considerable en el software de optimización. De particular interés es el uso de las redes neuronales que proporcionan gran rapidez y flexibilidad.

Por contra, también se han usado una serie de herramientas que no han mostrado el rendimiento esperado. El algoritmo *GlobalSearch*, que recordemos se encarga de encontrar el mínimo global de una función, ha tenido un rendimiento inapropiado en gran parte de los ejemplos. El optimizador pocas veces ha sido capaz de converger con este segundo solver, generando geometrías alejadas de la óptima.

Otra herramienta que no ha cumplido su función como se esperaba ha sido GiD. Por un lado, ha permitido generar geometrías que con Matlab sería imposible, pero, por otro lado, muchas de estas geometrías contenían errores serios que han impedido el posterior cálculo de elementos finitos.

6 TRABAJOS FUTUROS

Tras haber desarrollado el programa desde cero, haber hecho una gran cantidad de modificaciones y por supuesto, haber ejecutado un gran número de análisis, se plantean las siguientes mejoras y trabajos futuros que no han podido ser abordadas en el desarrollo de este TFG:

- El uso del procedimiento *GlobalSearch* no ha proporcionado los resultados esperados, no obteniéndose la convergencia en la mayor parte de ejemplos. La documentación que proporciona Matlab sobre esta subrutina dista de ser todo lo clara que sería necesario para comprender el funcionamiento del algoritmo. Como trabajo futuro se propone buscar información adicional sobre este algoritmo que permita parametrizarlo adecuadamente. Adicionalmente también se podría desarrollar un algoritmo alternativo que funcionase, al igual que *GlobalSearch,* bajo la idea de muestrear el espacio de soluciones a fin de maximizar la probabilidad de encontrar el mínimo global.
- Otra alternativa a lo planteado en el párrafo anterior es la de desarrollar un algoritmo híbrido de optimización que constaría de dos pasos. En un primer paso se usarían algoritmos evolutivos, que resultan muy sencillos y robustos y que realizan un amplio muestreo del espacio de soluciones, para determinar la zona donde se localiza el mínimo global. Después se utilizaría un algoritmo de tipo gradiente como *fmincon* para afinar la búsqueda del óptimo a partir de la solución proporcionada por el algoritmo evolutivo.
- Se ha visto que las redes neuronales artificiales ofrecen una respuesta muy precisa cuando se entrenan adecuadamente, pero se podría estudiar algunas alternativas y comparar el rendimiento de los diversos métodos. Los procesos estadísticos como la regresión estiman la relación existente entre variables independientes y la variable dependiente. El fin último es el de estimar una función y de esta forma obtener resultados instantáneos.

7 BIBLIOGRAFÍA

[1] *www.mathworks.com*

[2] *www.gidhome.com*

[3] *José Pascual Manzano, "Desarrollo de optimizador de forma 2D basado en código de elementos finitos con mallados cartesianos integrado con redes neuronales", Proyecto final de carrera, UPV, Junio de 2015.*

[4] *Onofre Marco, "A shape sensitivity analysis module with geometric representation by NURBS for a 2D finite element program based on cartesian meshes independent of geometry", Tesis de master, UPV, Julio de 2012.*

[5] *Josep Tornero y Leopoldo Armesto, "Técnicas de optimización", Editorial de la UPV, ISBN: 978-84-8363-191-1.*

[6] *Apuntes de la asignatura "Técnicas computacionales para Ingeniería Mecánica".*

Planos

Debido al carácter informático del proyecto, no procede la realización de planos.

Pliego de condiciones

1 INTRODUCCIÓN

El presente pliego de condiciones tiene por objetivo establecer las relaciones entre el proyectista, promotor del proyecto, usuario y demás partes que pudieran estar implicadas en este proyecto. Se describirán las condiciones generales del trabajo y se establecerá con el usuario final de la aplicación las condiciones que tendrán éste sobre la distribución y la utilización de la misma.

En este capítulo del proyecto, se señalan los derechos, obligaciones y responsabilidades de cada una de las partes involucradas.

2 CONDICIONES GENERALES

2.1 CONDICIONES LEGALES

Se reconoce públicamente las marcas registradas que aparecen en el desarrollo del proyecto, así como los derechos de autor de la bibliografía consultada para la realización del mismo.

No está permitida la reproducción total o parcial de este proyecto, ya sea el programa o la documentación técnica aportada, ni su tratamiento informático, ni la transmisión de ninguna forma o por cualquier medio sin el permiso previo y por escrito del ingeniero firmante del proyecto.

2.2 CONDICIONES ADMINISTRATIVAS

El promotor del proyecto es el Departamento de Ingeniería Mecánica y de Materiales de la Universidad Politécnica de Valencia con el profesor y tutor D. Juan José Ródenas García como representante del mismo y adscrito a la Escuela Técnica Superior de Ingeniería del Diseño de la misma universidad.

El proyectista es el alumno de Ingeniería Mecánica D. David Muñoz Pellicer, el cual está sometido a las siguientes obligaciones:

- Cumplir con la legislación vigente.
- Llevar a cabo el proyecto siguiendo las directrices del promotor.
- Cumplir con la normativa sobre la realización de trabajos de fin de grado vigente en la Universidad Politécnica de Valencia, la Escuela Técnica Superior de Ingeniería del Diseño y el Departamento de Ingeniería Mecánica y de Materiales.
- Respetar las leyes sobre derechos de autor.
- Consultar con el promotor cualquier modificación de las especificaciones iniciales, así como proponer soluciones alternativas a los problemas que puedan surgir.
- Informar periódicamente al promotor del estado de desarrollo del proyecto.

El proyectista estará amparado por una serie de derechos:

- Disponer de un equipo informático adecuado para la realización del proyecto.
- Ser informado por el promotor sobre los derechos legales acerca del proyecto.
- Recibir soporte técnico para cualquier problema imprevisto que pueda surgir.
- En caso de ausencia del promotor del proyecto, el proyectista tendrá plena potestad en la toma de decisiones relativas a la ejecución del proyecto, las cuales deberán ser asumidas por el promotor.

3 CONDICIONES TÉCNICAS PARTICULARES

Debido al carácter informático de este proyecto, se derivan una serie de condiciones y requisitos que limitan la utilización del mismo. El no cumplimiento de los requisitos no implica el mal funcionamiento de la aplicación, pero si una posible causa de éste.

3.1 EQUIPO FÍSICO DE TRABAJO

El *hardware* con el que se ha desarrollado el proyecto, y, por tanto, configuración recomendada, es la siguiente:

- Procesador Intel Core i7-4770 con 8 núcleos a 3.40 GHz.
- 16 GB de memoria RAM.
- Tarjeta gráfica NVIDIA GeForce GTX 780.
- Unidad de disco rígido (HDD) genérica.
- Monitor genérico.
- Teclado genérico.
- Ratón genérico.

Cualquier configuración de equipo que tenga características similares o que mejore alguno de sus aspectos también sería válido.

3.2 PROGRAMAS

Existen tres requisitos en cuanto a *software* que debe presentar el equipo en el que se vaya a ejecutar la aplicación, estos serán:

Sistema operativo:

- Microsoft Windows 7 o superior.

Programas instalados:

- MathWorks MatLab R2014a o superior.
- GiD 12.0.8 o superior.

3.3 CONDICIONES DE EJECUCIÓN Y LIMITACIONES DE RESPONSABILIDAD

La ejecución de la aplicación debe hacerse tal y como se especifica a lo largo de la memoria y en el Manual de Usuario (ver Anexo 1) que se adjunta en la documentación técnica, sin hacerse responsable el ingeniero firmante de la pérdida o daños en el *hardware* o *software* del ordenador donde se instale y ejecute el programa, provocados por el uso incorrecto de la aplicación, tanto en la instalación como en su posterior utilización.

Presupuesto

1 CUADRO DE PRECIOS UNITARIOS

Materiales

Ids.	Unidades	Descripción	Coste (€)
MT1	Uds	Equipo informático	650,00

Mano de obra

Ids.	Unidades	Descripción	Coste (€)
MO1	Horas	Ingeniero mecánico	60,00
MO2	Horas	Ingeniero industrial - Especialista	120,00

2 CUADRO DE PRECIOS DESCOMPUESTO

Trabajo 1

Ids.	Unidades	Descripción
ID01	Uds	Planteamiento del problema y definición de objetivos

Mano de obra directa

Ids.	Unidades	Descripción	Coste (€)	Cantidad	Parcial
MO1	Horas	Ingeniero mecánico	60,00	12	720,00
MO2	Horas	Ingeniero industrial - Especialista	120,00	3	360,00

Trabajo 2

Ids.	Unidades	Descripción
ID02	Uds	Desarrollo y diseño de la aplicación

Mano de obra directa

Ids.	Unidades	Descripción	Coste (€)	Cantidad	Parcial
MO1	Horas	Ingeniero mecánico	60,00	140	8400,00
MO2	Horas	Ingeniero industrial - Especialista	120,00	7	840,00

Trabajo 3

Ids.	Unidades	Descripción
ID03	Uds	Fase de prueba, depuración y finalización de la aplicación

Mano de obra directa

Ids.	Unidades	Descripción	Coste (€)	Cantidad	Parcial
MO1	Horas	Ingeniero mecánico	60,00	60	3600,00
MO2	Horas	Ingeniero industrial - Especialista	120,00	3	360,00

Trabajo 4

Ids.	Unidades	Descripción
ID04	Uds	Realización y preparación de la documentación técnica

Mano de obra directa

Ids.	Unidades	Descripción	Coste (€)	Cantidad	Parcial
MO1	Horas	Ingeniero mecánico	60,00	30	1800,00
MO2	Horas	Ingeniero industrial - Especialista	120,00	4	480,00

3 MEDICIÓN Y PRESUPUESTO

Mano de obra directa

Ids.	Unidades	Descripción	Coste (€)	Cantidad	Parcial
ID01	Uds	Planteamiento del problema y definición de objetivos	1080,00	1	1080,00
ID02	Uds	Desarrollo y diseño de la aplicación	9240,00	1	9240,00
ID03	Uds	Fase de prueba, depuración y finalización de la aplicación	3960,00	1	3960,00
ID04	Uds	Realización y preparación de la documentación técnica	2280,00	1	2280,00

Materiales

Ids.	Unidades	Descripción	Coste (€)	Cantidad	Parcial
MT1	Uds	Equipo informático	650,00	1	650,00

Medios auxiliares

Ids.	Unidades	Descripción	Coste (€)	Cantidad	Parcial
MA1	%	Medios auxiliares sobre costes directos	17210,00	12	2065,20

Presupuesto de ejecución material

Descripción	Subtotal (€)
Unidades de obra	16560,00
Materiales	650,00
Medios auxiliares	2065,20

El presupuesto de ejecución material asciende a **DIECINUEVE MIL DOSCIENTOS SETENTA Y CINCO EUROS CON VEINTE CÉNTIMOS (19275,20 €).**

Anexos

1 ANEXO: MANUAL DE USUARIO

Se detallará una visión general de los cambios que hay que realizar en los parámetros que definen el programa con el fin de ejecutar un análisis u otro en función de nuestras necesidades. Para apreciar con mayor claridad estos, la explicación se apoyará en una imagen de la función que rige la ejecución y que el usuario debe conocer. Esta función es la llamada a lo largo de la memoria *Optimization*.m, y se dividirá en partes.

```
% Adding path folders
Batch_Launcher
% Input of problem parameters
Batch_ProblemParameters(...
1   '08_Cylinder',... % Problem name.
    '08_Cylinder',... % Problem folder.
    'Optimization'); % Problem type.
% Input of analysis parameters
Batch_GeometryParameters(...
    {[]},... % Names of the files (.igs/.mat) to import. Cell array (1xNumBodies)
    {zeros(3,1)},... % Bodies movement. Cell array (1xNumBodies) {zeros(3,1)}
    {eye(3)},... % Bodies rotation. Cell array (1xNumBodies) %{eye(3)} {[0 0 1;0 1 0;1 0 0]}
    {'Dummy'},... % Name of the materials in the library. Cell array (1xNumMaterials)
    1e3,... % Young modulus of the materials. Vector (1xNumMaterials)
    0.3,... % Poisson ratio of the materials. Vector (1xNumMaterials)
    {1}); % Index of the material for each volume. Cell array (1xNumBodies(1xNumVolumes))
Batch_MeshingParameters(...
2   1,... % Element type, linear (1) or quadratic (2). Single value
3   5,... % Initial level of mesh, usually it is recommended to start from 2. Vector (1xNumBodies)
    {0.013*ones(2,3)},... % Scale off set for the mesh in part of unity, ([SupX,SupY,SupZ;InfX,InfY,
    1,... % Maximum number of iterations. Single value
    0,... % Error target. Single value
    0.5,... % Relative error reduction per iteration. Single value
    0,... % Refinement type, uniform (1) or h-adaptive (0). Single value
    0,... % Mesh coarsening, off (0) or  on (1).
    [1 1 1],... % Preprocessing refinement criteria flags. [Geometrical,Image,Singularities] Vector
    1,... % Refinement minimum level. Single value
    6); % Refinement maximum level, limited up to 15. Single value
Batch_ErrorEstimationParameters(...
    1,... % Exact solution case. If (0) there will be no comparisons. Vector (1xNumBodies)
    1); % Superconvergent Patch Recovery type. Vector (1xNumBodies)
Batch_BoundaryConditionsParameters(...
    {[1 1 0 0 0 1]},... % Load case info. Cell array (1xNumBodies) [Dirichlet,Neumann,Volumetric,Po
    2,... % Stresses type to feedback the Dirichlet stabilization, (1) FEM, (2) SPR or (3) Exact st
    100,... % Scale the factor for Dirichlet imposition. Vector (1xNumBodies)
    0,... % Flag to enable stabilization of Neumann boundaries
    0.0001);  % Scale the factor for for Neumann stabilization
Batch_IntegrationAndSolverParameters(...
    1,... % Volume and surface approximation, linear isoparametric (1) or isogeometric (2). Single
    [5,5,3,5,3],... % Integration points in isogeometric approx. [FacetR,FacetS,FacetT,EdgeR,EdgeT]
    1,... % Use of Nested Reordering
    2,... % Treatment of node constraints (MPCs and Dirichlet at nodes), (0) Everything explicit, (
    0); % Direct solver (0) or iterative solver (1)
%% Constants creation
Data_ConstantsCreation
Intg_MarchingCubesIntersectionPatternsCreation
Intg_StandardDelaunayTetrahedralizationsCreation
% Input of program parameters
Batch_ProgramParameters(...
4   1,... % Use of GPUs for faster calculations, off (0) or  on (1).
    0,... % Profiler report, off (0) or  on (1).
    1,... % Display results after the analysis, off (0) or  on (1).
    0,... % Export results to Excel after the analysis, off (0) or  on (1).
    0,... % Export results to GiD, off (0) or  on (1).
    0,... % Save the problem data, off (0), on (1) or sensitivities (2).
    0,... % Use of parallel CPU's with matlabpool
    0,... % Check FE matrix self adjointness
    1);   % Options on Result calculation: (0) Not Computed, (1) Full Size Mode, (2) Load Case Part
```

Figura 52. Optimization (I).

En la parte del código vista en la anterior imagen (*Figura 52*), aparece en gran medida el código de elementos finitos. A esta parte del código solo le faltaría la función que ejecuta el análisis para tener el programa al completo. Por lo tanto, para el usuario carece de interés conocer el funcionamiento de cada uno de sus apartados. En este caso concreto solo le haría falta conocer cómo cambiar el tipo de geometría que se está ejecutando, el tipo de los elementos, el tamaño de malla de elementos finitos y tal vez, la elección de usar GPUs o no para realizar los cálculos.

En el caso de querer cambiar la geometría (**1 en la imagen**), se debe modificar el nombre de la carpeta, existen algunos ejemplos predefinidos y se puede elegir cualquiera de ellos.

Si se desea cambiar el tipo de elemento (**2 en la imagen**), se deben seguir las siguientes instrucciones: para elementos lineales debe aparecer un 1 y si se desean elementos cuadráticos deberá aparecer un 2. El tamaño de malla se definiría a continuación (**3 en la imagen**). En función de la precisión y el tiempo de ejecución deseada se seleccionará el tamaño de malla adecuado. Por último, si el ordenador que se usa para el análisis no dispone de GPUs o si se prefieren ejecutar los cálculos en la CPU se deberá cambiar el valor que aparece a 0 (**4 en la imagen**).

A continuación se verá el código que define los parámetros del módulo de sensibilidades (*Figura 53*).

```
% Sensitivities parameters definition
Batch_SensitivitiesParameters(...
    'Elastic',... % Analysis type. Volume derivatives ('Geometry') or elastic derivatives ('Elastic'
1   5,... % Design variables case. <-----
    0,... % Exact solution case. If (0) there will be no comparisons.
2   [15 15 15 15 15 15 15 15],... % Design parameters. <-----
3   2,... % Velocity field type. (1) analytical, (2) physical, (3) physical on boundary, (40,41,42)
    [],... % Singularity index
    0,... % Target mesh, index of the target mesh, (0) last mesh.
    [1 1],... % Stabilization stresses for [VelocityField,SensitivityAnalysis]
    1,... % Use of GPUs for faster calculations, off (0) or  on (1).
    0,... % Iterative solver
    0); % Save the problem data, off (0), on (1).
```

Figura 53. Optimization (II).

Para cambiar el caso de diseño, se deberá cambiar el valor del parámetro que aparece marcado como **1 en la imagen**, siguiendo las directrices mostradas a lo largo de la memoria. Si se modificara el caso de diseño es posible que también se deban modificar los parámetros de diseño (**2 en la imagen**), se ha de suministrar un escalar o un vector de un tamaño acorde con el caso de diseño, estos valores definirán la geometría inicial. Por último, es posible modificar el campo de velocidades (**3 en la imagen**), la elección de éste tiene una relación directa con la precisión de los cálculos de sensibilidades de forma. Este parámetro se puede modificar y ver los resultados, pero es posible que se necesiten algunos conocimientos adicionales para comprender completamente su funcionamiento.

La siguiente porción del código de *Optimization* será la encargada de definir los parámetros referentes a la optimización (*Figura 54*).

```
% Optimization parameters definition
OptimParameters(...
1   0,... % Use of Neural Network, off (0) or on (1).
    1,... % Use of Storage Data, off (0) or on (1).
    0,... % Use of Previous Analysis, off (0) or on (1).
2   5,... % Design case. <-----
    [15 15 15 15 15 15 15 15],... % x0, value to inicialize the optim problem (Design parameters).
    0,... % ConstantMesh.
3   10,... % NumTrialPoints, number of points to inicialize GlobalSearch.
    10,... % NumStageOnePoints, number of points evaluated in the first stage.
4   'sqp',... % Algorithm, 'interior-point', 'sqp', 'trust-region-reflective', 'active-set'.
    'final',... % Display, 'final', 'iter', 'off'.
5   1e-4,... % TolFun, tolerance of function to optimize.
    1e-4,... % TolX, tolerance of inputs in the optimization analysis.
    1e-4,... % TolCon, tolerance of restriction.
    'never');... % UseParallel
```

Figura 54. Optimization (III).

En esta parte del script el usuario puede cambiar cualquiera de los parámetros, pero se detallarán los más importantes:

Los primeros tres valores (**1 en la imagen**) hacen referencia a la ejecución o no de alguna de las mejoras. Con el primero de ellos se ejecutarían las redes neuronales, con el segundo el programa recuperaría los resultados almacenados y con el tercero el programan recuperaría adicionalmente los resultados de una ejecución anterior. Cada una de estas características se activaría escribiendo un 1, mientras que para la desactivación se sustituiría por un 0.

Los parámetros que aparecen a continuación (**2 en la imagen**), son los encargados de definir el caso de diseño y las variables de diseño, se modifican de acuerdo al apartado anterior.

Los siguientes dos parámetros (**3 en la imagen**) rigen en gran medida el funcionamiento de *GlobalSearch*, en el primero de ellos se selecciona la cantidad de puntos por los que iniciará el algoritmo, mientras que el segundo define cuántos se deben ejecutar en una primera etapa.

Los dos parámetros que aparecen a continuación (**4 en la imagen**) sirven para definir, por un lado, el algoritmo que debe seguir el optimizador para obtener los nuevos puntos de ejecución y, por otro lado, la forma en la que se quiere visualizar la ejecución de la optimización. Ambos parámetros se pueden modificar siguiendo las instrucciones que aparecen a su derecha. En estas instrucciones aparecen todos los posibles valores que pueden adquirir estos parámetros.

El último grupo de parámetros (**5 en la imagen**), hace referencia a las tolerancias que aparecen en la ejecución del programa, estas tolerancias definirán en gran medida el criterio de convergencia. Las tolerancias a modificar son: la de la función objetivo (*TolFun*), la de la restricción (*TolCon*) y la de las variables de diseño (*TolX*).

En la siguiente imagen se muestra el código que ejecuta el programa (*Figura 55*) y solo se muestra porque en función del valor de entrada que el usuario decida introducir éste se ejecutará de una forma o de otra. En caso de seleccionar "**Global**", se buscará el mínimo global, mientras que si se introduce "**Local**" el optimizador convergerá en el primer mínimo local.

```
xmin = OptimizationFunction('Global');% Switch 'Local' or 'Global' to define the type of optimizati
```

Figura 55. Optimization (IV).

Es posible que el usuario decida modificar algún parámetro que no esté definido en el script anterior.

En caso de que el usuario quiera cambiar las restricciones de las variables de diseño el usuario tendrá que acceder a *OptimParameters*. Ésta función es accesible desde *Optimization* y se podrá modificar cualquier valor que aparezca en la *Figura 56*.

```
switch numel(x0)
    case 1
        lb = 6;
        ub = 25;
        Aineq = [];
        bineq = [];
    case 2
        lb = [9 7];
        ub = [25 22];
        Aineq = [0.75 -1];
        bineq = 0.5;
    case 4
        lb = [9 7 7 9];
        ub = [16 15 15 16];
        Aineq = [0.75 -1 0 0;0 1 -1 0;1 0 0 -1;-1 0 0 1];
        bineq = [0.5;0.1;0.1;0];
    case 8
        lb = [9 7 7 9 9 7 7 9];
        ub = [16 15 15 16 16 15 15 16];
        Aineq = [0.75 -1 0 0 0 0 0 0;0 1 -1 0 0 0 0 0;1 0 0 -1 0 0 -0 0;-1 0 0 1 0 0 0 0;...
            0 0 0 0 0.75 -1 0 0;0 0 0 0 0 1 -1 0;0 0 0 0 1 0 0 -1;0 0 0 0 -1 0 0 1];
        bineq = [0.5;0.1;0.1;0;0.5;0.1;0.1;0];
end
Aeq = []; beq = [];
```

Figura 56. Restricciones en OptimParameters.

En esta parte de la función se selecciona unas restricciones u otras según el valor de las variables de diseño. Antes de modificar cualquier valor, se deberá tener en cuenta que las restricciones están en formato matricial y que cumplen la siguiente regla:

$$Aineq \cdot x \leq bineq$$

El último de los apartados que el usuario debe conocer es el que define en gran medida la generación de las redes neuronales, así como la definición de la restricción entre otros. Estos parámetros están definidos al comienzo de la función *OutputObtention* y el usuario deberá acceder a ésta para modificar alguno de estos parámetros (*Figura 57*).

```
global Iteration ResultData Analysis NewPoints NNet NNetUses RepPos Parameters
TensAdm = 2;
FeedNNetResults = 30;
MaxNNetUses = 100;
OptimLevel = 3;
```

Figura 57. Parámetros en OutputObtention.

Cualquiera de estos parámetros es ampliamente modificable con el fin de observar las distintas ejecuciones posibles. El usuario puede variar el valor de la máxima tensión admisible, la cantidad de datos que necesitan las redes para ser entrenadas adecuadamente, la cantidad de veces que se pueden ejecutar estas redes antes de que se queden obsoletas o la selección del tamaño óptimo de malla.

Estos parámetros se han definido dentro de la propia función con el fin de eliminar el mayor número de variables globales posible, ya que estas consumen más recursos que las variables locales.

Una vez ya se hayan definido todos los parámetros a preferencia del usuario, se procederá a realizar el análisis ejecutando el script *Optimization*.m. Cuando el proceso concluya aparecerá por pantalla una de las dos figuras siguiente (*Figura 58 y Figura 59*) en función de las condiciones definidas.

```
Local minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the selected value of the optimality tolerance,
and constraints are satisfied to within the selected value of the constraint tolerance.

<stopping criteria details>
```

Figura 58. Mensaje final de la ejecución local.

En el caso de la ejecución global, aparecería el mensaje anterior tantas veces como solvers se hayan ejecutado junto con el siguiente texto:

```
GlobalSearch stopped because it analyzed all the trial points.

All 2 local solver runs converged with a positive local solver exit flag.
```

Figura 59. Mensaje final de la ejecución global.

Se han implementado unas líneas de código que generan un archivo de texto, en el que aparecen los resultados más relevantes del análisis con el fin de comprobar el estado de la ejecución. El formato de este texto sería el que aparece en la *Figura 60*:

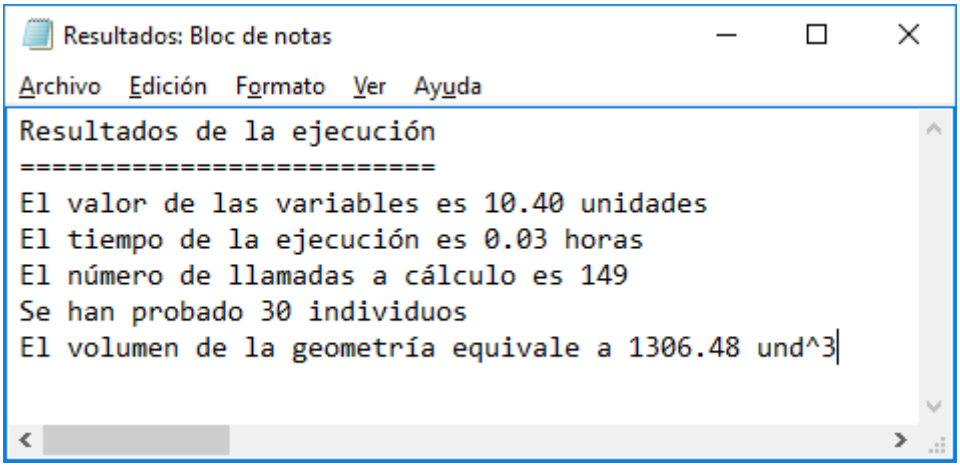

Figura 60. Informe final.