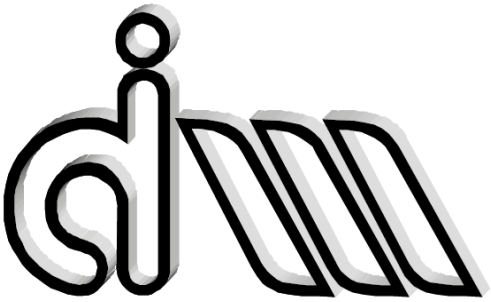

UNIVERSIDAD POLITÉCNICA DE VALENCIA

Escuela Técnica Superior de Ingeniería del Diseño

PROYECTO FINAL DE CARRERA

Ingeniería Aeronáutica

DESARROLLO DE OPTIMIZADOR DE FORMA 2D BASADO EN CÓDIGO DE ELEMENTOS FINITOS CON MALLADOS CARTESIANOS INTEGRADO CON REDES NEURONALES

Autor: José Pascual Manzano Pérez

Directores: Juan José Ródenas García

José Albelda Vitoria

Valencia, Junio de 2015

AGRADECIMIENTOS

A mis padres, por haberme animado en los momentos más difíciles.

A mis directores de proyecto, Juan José Ródenas García y José Albelda Vitoria, por guiarme durante la elaboración del mismo y estar disponibles todas las semanas.

Al Departamento de Ingeniería Mecánica y de Materiales (DIMM), por permitirme el uso de uno de sus ordenadores para la realización de los cálculos del proyecto.

A Onofre, por su rapidez y claridad a la hora de resolver mis dudas.

RESUMEN

La optimización de forma es una técnica matemática utilizada para resolver problemas que consisten en encontrar el contorno que minimice un cierto coste funcional satisfaciendo unas determinadas restricciones.

La optimización de forma de componentes estructurales requiere la realización de muchos análisis numéricos, lo que conlleva un alto coste computacional que en ocasiones puede incluso disuadir del uso de estas técnicas. La generalización del uso de técnicas de optimización en la industria requiere, pues, el desarrollo de procedimientos numéricos computacionalmente eficientes. Se pueden seguir dos caminos con esa finalidad: el desarrollo de software de análisis mediante el método de los elementos finitos más eficientes y el desarrollo de algoritmos de optimización más eficientes.

Este proyecto es la continuación del desarrollo del software de optimización de forma 2D del Departamento de Ingeniería Mecánica y de Materiales de la Universidad Politécnica de Valencia, con el objetivo de mejorarlo en varios aspectos: facilitar su uso al usuario, su edición al programador, y reducir su tiempo de ejecución mediante el desarrollo de un algoritmo de optimización más eficiente.

Los resultados obtenidos son sorprendentes; cumplen con creces todas las expectativas.

ESTRUCTURA DE LA MEMORIA

La memoria está estructurada de la siguiente forma:

- En el capítulo “Introducción” se presenta todo el contenido teórico que el lector necesita tener en cuenta para situarse en el marco del proyecto y se exponen los objetivos del mismo y las soluciones propuestas.
- En el capítulo “Análisis” se define el tipo de problema que el software desarrollado debe resolver.
- En el capítulo “Diseño” se describe el funcionamiento del software desarrollado.
- En el capítulo “Resultados” se presentan los resultados obtenidos tras la ejecución de varios ejemplos y se evalúan según los objetivos planteados.
- En el capítulo “Conclusiones y trabajos futuros” se analiza la consecución final de los objetivos del proyecto y se plantean posibles mejoras.
- En los apéndices del proyecto están el manual del usuario del software desarrollado y el manual del programador, donde se explica el código del programa en detalle.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. INTRODUCCIÓN

La optimización de forma es una técnica matemática utilizada para resolver problemas que consisten en encontrar el contorno que minimice un cierto coste funcional satisfaciendo unas determinadas restricciones.

La utilización de técnicas de optimización de forma en la industria conduce a la obtención de piezas de menor peso y mejores prestaciones. La disminución del material usado en una pieza suele conllevar una reducción de costes importante en sectores donde se fabrican grandes series.

Los algoritmos basados en el gradiente buscan el valor *x* que corresponda al mínimo de una función *f(x)*, con las direcciones de búsqueda definidas por el gradiente de la función en el punto actual de ésta.

Los procesos de optimización basados en el gradiente se constituyen por un nivel "superior" gobernado por el algoritmo de optimización y un nivel "inferior" gobernado por el método numérico utilizado para analizar cada solución propuesta. En problemas relativos a la optimización de forma de componentes mecánicos estructurales se ha de utilizar, en el nivel inferior, un método numérico como el método de los elementos finitos (MEF) para determinar el valor de la función objetivo (masa, desplazamiento,…) y el grado de cumplimiento de las restricciones (tensiones máximas,…) con un nivel de precisión adecuado que garantice la convergencia del proceso.

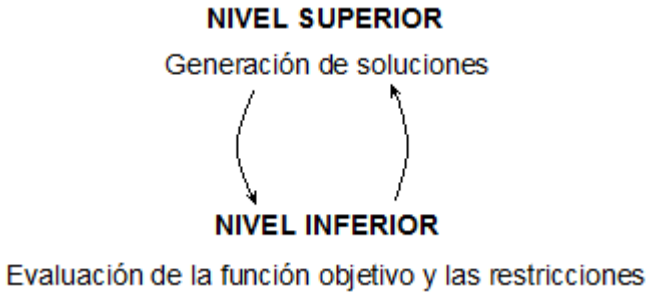

Figura 1. Niveles del proceso de optimización.

Dado que el proceso de optimización es iterativo, resulta necesario analizar múltiples geometrías. Los análisis MEF requieren un elevado esfuerzo computacional; el coste de los análisis de todas esas geometrías hace que el tiempo de ejecución del proceso de optimización se eleve mucho. El nivel "inferior" del proceso de optimización es, por tanto, responsable de la mayor parte del coste computacional.

Con el fin de reducir dicho coste, en este proyecto se plantea la utilización de redes neuronales artificiales para una evaluación aproximada (y prácticamente inmediata) de las funciones objetivo y del grado de cumplimiento de las restricciones del problema, utilizando una base de datos de soluciones previamente evaluadas mediante análisis MEF. El nivel "inferior" del proceso se basa, por tanto, en la realización de análisis MEF y la utilización de redes neuronales artificiales.

En este capítulo se desarrolla el marco teórico del proyecto y los antecedentes del mismo, así como el objetivo que se pretende alcanzar y las soluciones propuestas para ello.

1.1. MARCO TEÓRICO

Se procede a la exposición de un breve desarrollo teórico de las unidades temáticas en las que se basa la realización de este proyecto.

1.1.1. Método de los elementos finitos

Este apartado se ha obtenido de los apuntes de la asignatura "Cálculo Estructural, Método de los Elementos Finitos" [3]*, con el beneplácito de uno de sus autores (que es director de este proyecto).*

El método de los elementos finitos (MEF) es un método numérico de propósito general para la aproximación de soluciones de ecuaciones diferenciales parciales muy utilizado para la resolución de problemas de contorno que aparecen en la ingeniería.

Gracias al MEF, los problemas no lineales pueden tratarse más fácil y eficazmente, y es posible resolver problemas combinados que anteriormente se trataban por separado (análisis de tensiones, de temperaturas, de fluidos, etc.).

Como la solución alcanzada es una aproximación, son necesarias la estimación del error introducido y la adaptación del modelo numérico en base a dicha estimación de error, para lo cual existen técnicas fiables y rápidas que permiten garantizar un error menor que el deseado sin que el usuario tenga que intervenir.

La tendencia es automatizar todo el proceso de análisis para que el usuario pueda dedicar casi todo su tiempo a la interpretación de resultados.

En general se utilizan funciones de interpolación polinómicas en la formulación del MEF, lo que facilita los cálculos numéricos y el aumento de la precisión de los resultados (incrementando el orden del polinomio de interpolación).

Casi todos los elementos finitos son geométricamente simples para facilitar el modelado de cualquier dominio. Hay elementos unidimensionales, bidimensionales (los más comunes son los triangulares y los cuadriláteros) y tridimensionales (tetraédricos, hexaédricos y prismas triangulares). A su vez, éstos pueden ser lineales (polinomio de interpolación de orden uno), cuadráticos (polinomio de interpolación de orden dos), cúbicos (polinomio de interpolación de orden tres), etc.

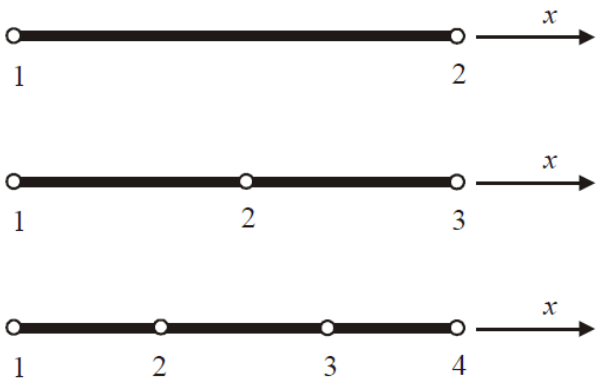

Figura 2. Elementos unidimensionales [3].

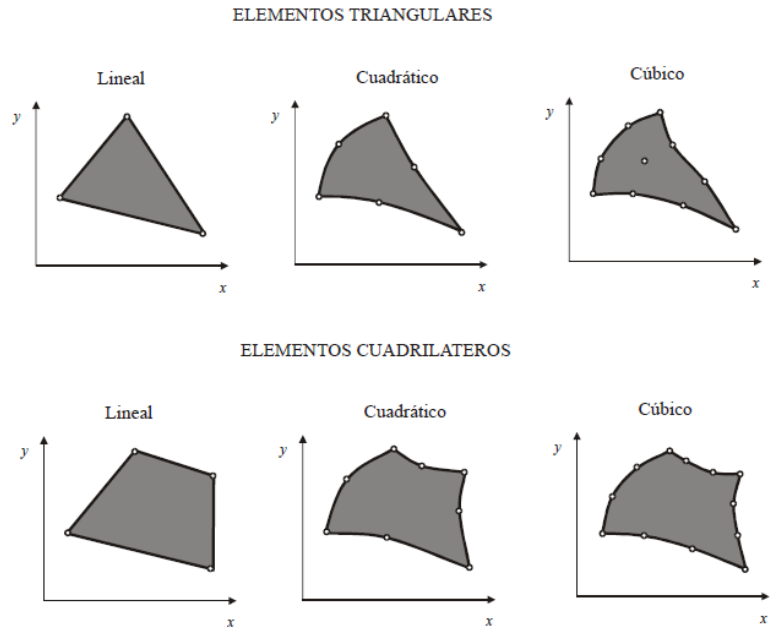

Figura 3. Elementos bidimensionales [3].

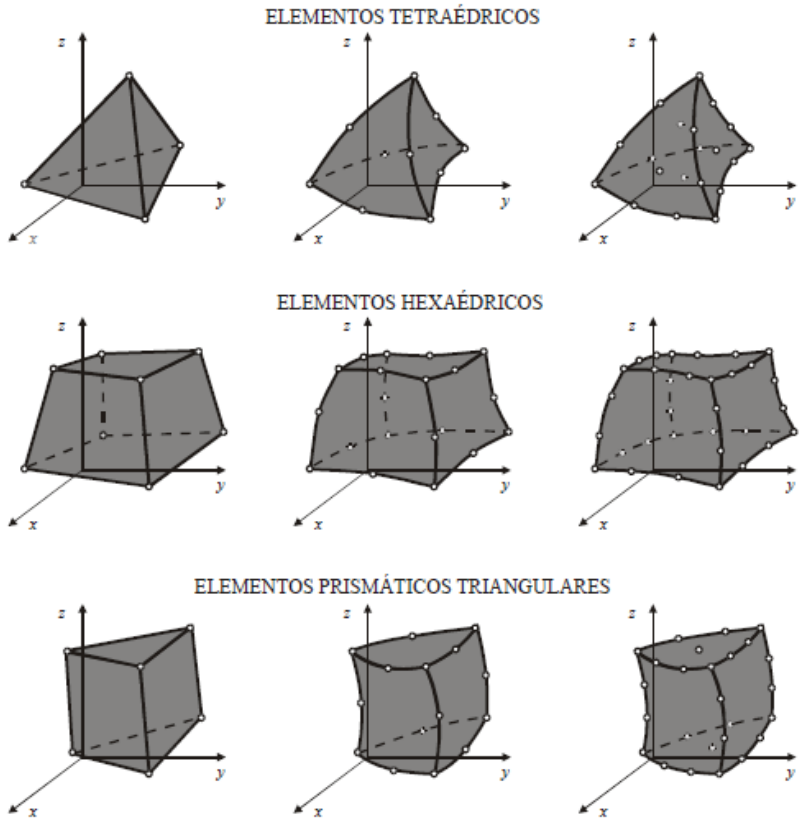

Figura 4. Elementos tridimensionales [3].

El error entre la solución exacta de un problema y la obtenida mediante el MEF está influenciado por varios tipos de errores:

- Errores de modelado, dada la diferencia entre el sistema físico y su modelo matemático.
- Errores de redondeo, al utilizar un número limitado de dígitos para representar números reales.
- Errores de manipulación, introducidos por los algoritmos utilizados.
- Errores de integración, al usar un método numérico aproximado.
- Errores de discretización, debidos a la representación de los infinitos grados de libertad de un continuo mediante un número finito de ellos.

El uso de la doble precisión y algoritmos de cálculo apropiados hace que los errores de redondeo y manipulación sean despreciables. Mediante el refinamiento de la malla, los errores de modelado se reducen tanto que la fuente más importante de error en los análisis MEF es el error de discretización.

Para cuantificar ese error se utiliza una magnitud llamada *norma energética*. La norma energética del error de discretización es la raíz cuadrada del doble de la energía de deformación del error. Para estimarlo se utiliza el estimador de error de Zienkiewicz y Zhu, llegando a la expresión:

$$\eta_{es} = \frac{\|\boldsymbol{e}_{es}\|}{\sqrt{\left\|\boldsymbol{u}_{ef}\right\|^2 + \|\boldsymbol{e}_{es}\|^2}} * 100$$

Donde η_{es} representa la estimación del error relativo en norma energética en términos porcentuales, $\|\boldsymbol{e}_{es}\|$ es la estimación del error en norma energética y $\left\|\boldsymbol{u}_{ef}\right\|$ es la norma energética de la solución de elementos finitos.

Las técnicas adaptativas son procedimientos que disminuyen el error de discretización consistentes en mejorar el modelo de elementos finitos en las zonas donde éste proporciona mayor error. Las tres principales estrategias son:

- Procedimientos h-adaptativos, aquellos en los que se varía el tamaño de los elementos manteniendo el grado de interpolación.
- Procedimientos p-adaptativos, donde se varía el grado de los polinomios de interpolación manteniendo el tamaño de los elementos.

- Procedimientos hp-adaptativos, en los que se varían el tamaño de los elementos y el grado de interpolación.

Si bien la técnica más efectiva es el procedimiento hp-adaptativo, la más extendida es la del procedimiento h-adaptativo, utilizada en este proyecto.

1.1.2. Matlab

Matlab es un software matemático que ofrece un entorno de desarrollo integrado con un lenguaje de programación propio [11].

Permite la manipulación de matrices, la representación de datos y funciones, la implementación de algoritmos, la creación de interfaces de usuario (GUI) y la comunicación con programas en otros lenguajes y con otros dispositivos hardware.

1.1.3. Redes neuronales

Las redes neuronales artificiales son modelos matemáticos utilizados en diversas disciplinas que tratan de emular el funcionamiento de las redes neuronales biológicas.

Se trata de un sistema de “neuronas” interconectadas que colaboran entre sí para producir una determinada salida. Permiten modelar un problema concreto mediante un aprendizaje automático de las propiedades de éste.

El bloque fundamental para la construcción de redes neuronales es, pues, la neurona con una entrada (escalar o vectorial):

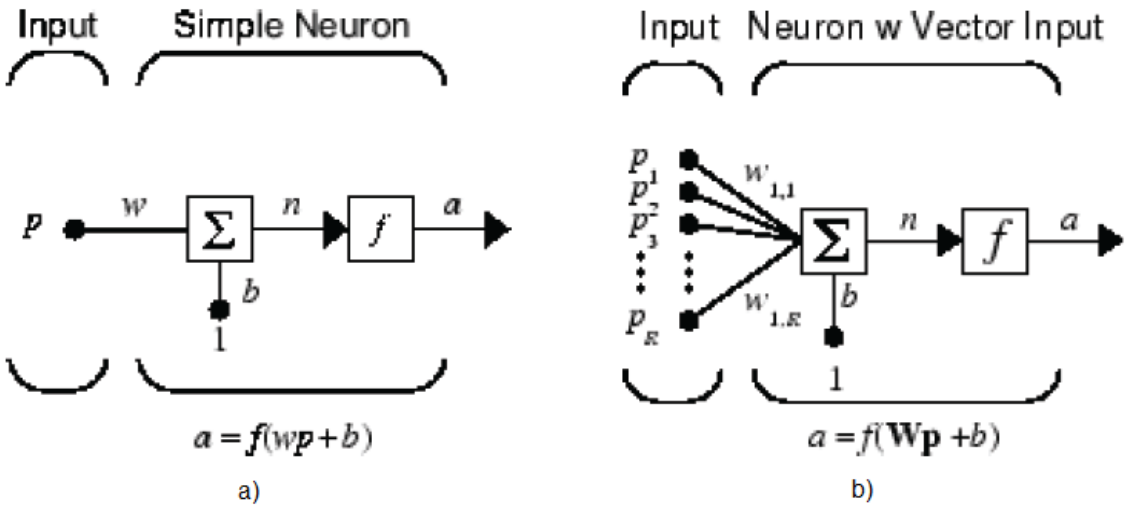

Figura 5. Modelo de neurona con una entrada escalar (a) y vectorial (b) [4].

En el modelo a) el valor de entrada escalar *p* es multiplicado por el peso escalar *w*, formando el producto escalar *wp*. Dicho producto es sumado junto a la tendencia (sesgo estadístico) *b* para formar la entrada de la red *n*. Finalmente, *n* pasa por la función de transferencia *f* y se produce la salida *a*.

Las funciones de transferencia más comunes son la lineal, la log-sigmoide y la tan-sigmoide.

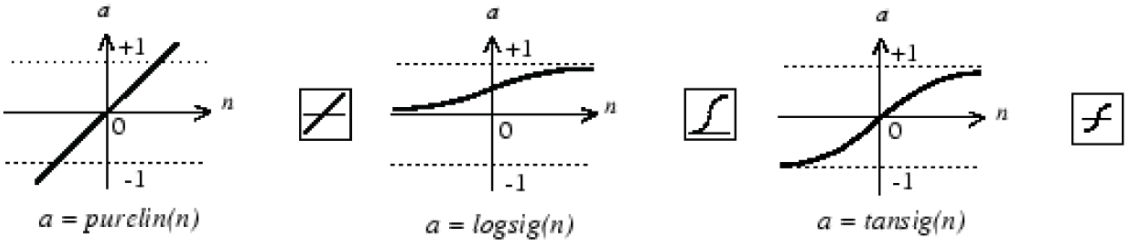

Figura 6. Funciones de transferencia más comunes [4].

Dos o más neuronas se pueden combinar en una capa. Una red neuronal puede contener una o más capas.

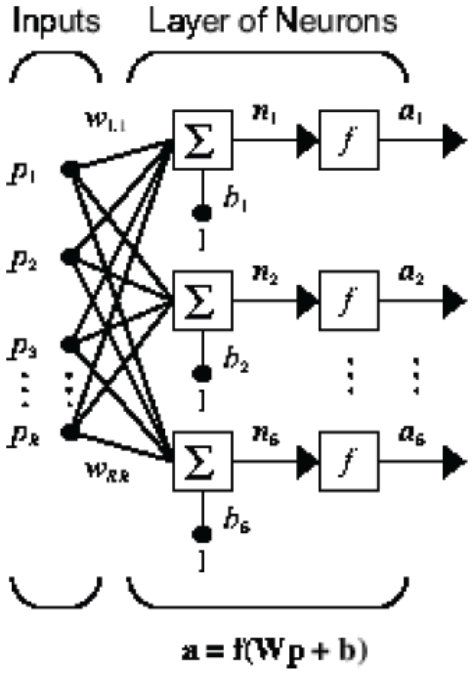

Figura 7. Modelo de una capa de neuronas [4].

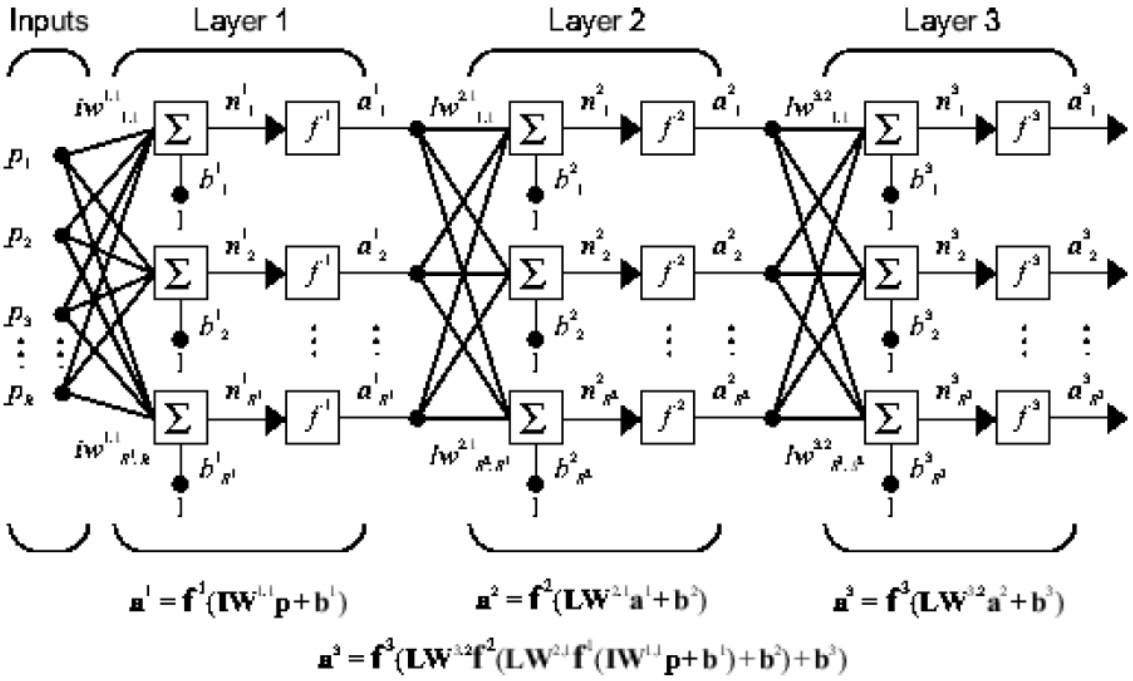

Figura 8. Modelo de múltiples capas de neuronas [4].

Las redes neuronales se pueden clasificar como estáticas o dinámicas. Las estáticas no poseen elementos de realimentación ni retrasos; la salida es calculada directamente utilizando la entrada a través de conexiones no cíclicas. En las dinámicas, la salida depende no sólo de la entrada presente, también de las entradas, salidas y estados de la red pasados. Las redes neuronales dinámicas son muy utilizadas en sistemas de control.

En este proyecto se usan redes neuronales estáticas.

El flujo de trabajo para toda red neuronal es el siguiente:

- Recopilar datos.
- Crear la red.
- Configurar la red.
- Inicializar pesos y tendencias.
- Entrenar la red.
- Validar la red.
- Utilizar la red.

1.2. ANTECEDENTES

La optimización de forma ya ha sido implementada en software: ANSYS, FreeFem++, ECS, MSC, etc. son buenos ejemplos de ello.

El Departamento de Ingeniería Mecánica y de Materiales (DIMM) de la Universidad Politécnica de Valencia lleva años desarrollando software encaminado a la optimización de forma, en 2D y 3D, programado en Matlab. El software desarrollado incluye un programa de elementos finitos (llamado cgFEM) que implementa la teoría del Método de los Elementos Finitos Generalizado y utiliza mallas cartesianas independientes de la geometría, lo que permite reducir el coste computacional necesario en los procesos de optimización.

Este proyecto es la continuación del desarrollo del software de optimización de forma 2D del DIMM. El software actual, resultado del trabajo de Luis Esparcia García [2], se utiliza como punto de partida.

Éste representa una implementación básica que muestra la viabilidad de la utilización de cgFEM como herramienta de análisis numérico en procesos de optimización de forma de componentes estructurales. Consiste en un conjunto de tres módulos principales de código para cada problema a tratar donde los datos particulares de cada problema se definen entre las mismas líneas de código del programa (de ahí que sea necesario un conjunto de módulos por problema), con algunos comandos redundantes, con un tiempo de ejecución dependiente por completo del tiempo de realización de análisis de elementos finitos (MEF), sin un proceso de tratamiento de posibles errores, con una cantidad insuficiente de información proporcionada al usuario y un inexistente aprovechamiento de los datos calculados para su potencial uso futuro.

1.3. OBJETIVO

El objetivo del proyecto es mejorar el actual programa de optimización de forma 2D del DIMM: facilitar su uso al usuario, su edición al programador, y reducir su tiempo de ejecución.

1.4. SOLUCIÓN PROPUESTA

Para mejorar el programa se proponen las modificaciones y los añadidos siguientes:

- Creación de un único conjunto de tres módulos principales donde se encuentra el código del programa, y un módulo adicional (necesario para cada problema que se desee resolver) donde se definen los datos del problema en cuestión.
- Organización de cada módulo en “bloques” con propósitos concretos y eliminación de las líneas de código innecesarias.
- Desarrollo de un proceso de tratamiento de posibles errores para evitar la interrupción del programa y proporcionar la información de las causas de dichos errores al usuario.
- Implementación de instrucciones para proporcionar al usuario una gran cantidad de información útil.

- Inclusión de un procedimiento con el fin de almacenar en disco la información ya calculada, lo que permite su utilización en futuras ejecuciones del programa y contribuye a la robustez de éste; si hay algún problema (por ejemplo, la falta de suministro eléctrico) una nueva ejecución puede continuar desde los últimos datos almacenados en la anterior.
- Adición del uso de datos almacenados (para evitar la repetición de cálculos) y redes neuronales (construidas utilizando información evaluada mediante análisis MEF) para acelerar el análisis de las soluciones propuestas (nivel “inferior” del proceso de optimización), disminuyendo así el tiempo de ejecución del programa.

2. ANÁLISIS

El problema general de optimización consiste en encontrar los valores de las variables de diseño del problema que se correspondan con los mínimos de unas funciones objetivo teniendo en cuenta unas restricciones determinadas.

Se puede distinguir entre problemas de optimización mono-objetivo (hay una única función objetivo) y multi-objetivo (hay varias funciones objetivo).

Los algoritmos de optimización se pueden clasificar siguiendo varios criterios:

- Algoritmos deterministas (producen los mismos resultados si se inician en un mismo punto del espacio de trabajo) o estocásticos (distintas ejecuciones de los algoritmos producen resultados diferentes).
- Algoritmos basados en el gradiente (requieren información del gradiente de la función objetivo) o no basados en el gradiente (necesitan sólo el valor de la función objetivo).
- Algoritmos de solución única (buscan futuras soluciones basándose en una única solución) o de solución múltiple (trabajan con varias soluciones a la vez).

Dentro de los algoritmos deterministas se tienen:

- Métodos no basados en el gradiente:
  - Métodos de “ascenso de colinas”.
  - Estrategia Simplex.
  - Estrategia coordinada.
- Métodos basados en el gradiente:
  - Gradiente conjugado (primera derivada).
  - Métodos de Quasi-Newton (primera derivada).
  - Método de Newton (segunda derivada).
  - Levenberg-Marquardt (segunda derivada).

Los métodos deterministas tienen la ventaja (normalmente) de una convergencia rápida. Sin embargo, convergen hacia el óptimo local más próximo, que en la mayoría de los casos no será el óptimo global. Por tanto, deben ser reiniciados varias veces, utilizando distintos puntos de inicio, para encontrar varios óptimos.

En el software desarrollado se utiliza un algoritmo determinista basado en el gradiente. Entre dichas técnicas, hay algunas que necesitan información de la primera derivada de la función objetivo, y otras que requieren información de la primera y la segunda derivada. Como el programa que realiza los cálculos de la función objetivo y las restricciones (cgFEM) tiene implementado el cálculo de las primeras derivadas únicamente, se ha seleccionado un algoritmo que usa solamente esa información: Quasi-Newton.

Para implementar todo esto en Matlab es necesaria la utilización del algoritmo *fmincon*, empleado en problemas de minimización:

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

Figura 9. Descripción del algoritmo *fmincon* [9].

b y *beq* son vectores, *A* y *Aeq* son matrices, *c(x)* y *ceq(x)* son funciones cuyas salidas son vectores, y *f(x)* es una función cuya salida es un escalar. *f(x)*, *c(x)* y *ceq(x)* pueden ser funciones no lineales.

x, *lb* y *ub* son escalares o vectores.

En los problemas que el programa tiene que resolver, se busca la minimización del volumen (que en el caso de un único material, es equivalente a minimizar la masa) de material utilizado en el componente a optimizar considerando restricciones de tensiones, desplazamientos, etc. De esta manera se tiene que:

- *x* es el vector de las variables de diseño del problema (las coordenadas de determinados puntos de la geometría que se quiere analizar). Si sólo hay una variable de diseño, *x* es un escalar.
- *f(x)* es el volumen del componente a optimizar.

- `c(x)` es el vector que contiene las restricciones de desigualdad del problema (máxima tensión de Von Mises-límite de fluencia, desplazamiento admisible de un punto-desplazamiento evaluado del punto, etc.).
- `ceq(x)` es el vector que contiene las restricciones de igualdad del problema (si las hubiera).
- *A* es una matriz que junto a *b*, que es un vector, contiene la información de las restricciones de desigualdad entre las variables de diseño.
- *Aeq* es una matriz que junto a *beq*, que es un vector, contiene la información de las restricciones de igualdad entre las variables de diseño (si las hubiera).
- *lb* y *ub* son vectores que contienen los límites inferiores y los superiores de los valores de las variables de diseño, respectivamente. Si sólo hay una variable de diseño, *lb* y *ub* son escalares.

Se han considerado otros algoritmos similares a *fmincon*: *fgoalattain* y *fminimax*, empleados en problemas multiobjetivo.

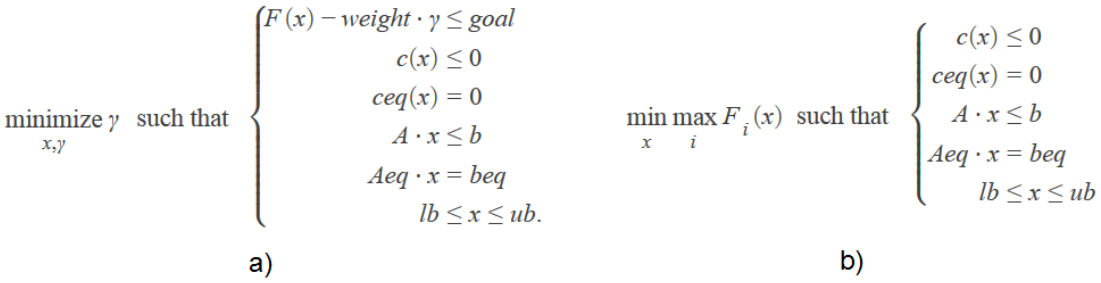

Figura 10. Descripción de los algoritmos *fgoalattain* (a) [8] y *fminimax* (b) [10].

Dichos algoritmos, sin embargo, tienen las siguientes desventajas:

- Pueden incumplir la restricción de los valores límite de las variables (*lb*≤*x*≤*ub*) en iteraciones intermedias del proceso.
- Las funciones con las que trabajan son asumidas continuas.
- Es posible que den únicamente soluciones locales.
- No pueden trabajar con el algoritmo *GlobalSearch*.

Lo que hace que *fgoalattain* y *fminimax* sean descartados.

El algoritmo *GlobalSearch* permite la búsqueda de soluciones desde múltiples puntos de inicio. Contiene propiedades que afectan a la forma en que el proceso de optimización busca un mínimo global. Las más significativas son:

- Número de puntos de inicio potenciales a examinar además del punto sugerido por el usuario (*NumTrialPoints*). *GlobalSearch* sólo trabaja con los que pasan unas pruebas determinadas.
- Número de puntos de inicio en la etapa 1 del algoritmo (*NumStageOnePoints*).
- Tipos de puntos de inicio que no se deben analizar (*StartPointsToRun*).

El proceso de *GlobalSearch* es el siguiente:

- Ejecuta *fmincon* desde el punto sugerido por el usuario.
- Genera los puntos de prueba (puntos de inicio potenciales).
- Etapa 1: ejecuta *fmincon* desde el mejor punto de inicio entre los primeros *NumStageOnePoints* puntos de prueba.
- Etapa 2: pasa por los puntos de prueba restantes y ejecuta *fmincon* si un punto determinado satisface las condiciones adecuadas.
- Crea el vector *GlobalOptimSolutions*.

De esta forma se consigue que el proceso de optimización busque los mínimos de la función deseada desde distintos puntos, lo que facilita la búsqueda del mínimo global.

En el código desarrollado, las propiedades de *GlobalSearch* se almacenan en la variable *gs*.

Las opciones del proceso de optimización se definen con la instrucción *optimset*. Las más significativas son:

- Algoritmo que usará el solver *fmincon* para encontrar los mínimos de la función (*interior-point*, *sqp*, *active-set* o *trust-region-reflective*).
- Utilización de gradientes proporcionados por el usuario (*GradObj*, *GradConstr*).
- Tolerancias y criterios de parada (*TolX*, *TolFun*).

En el código desarrollado, las opciones definidas en *optimset* se almacenan en la variable *opts*.

La estructura del proceso de optimización se define con la instrucción *createOptimProblem*. Aquí se definen:

- Solver utilizado (para trabajar con *GlobalSearch* debe ser necesariamente *fmincon*).
- La función objetivo del problema *f(x)*.
- Las funciones que gobiernan las restricciones del problema *c(x)≤0*, *ceq(x)=0*.
- El punto de inicio definido por el usuario *x0*.
- Las restricciones de igualdad y desigualdad entre las variables de diseño *Aeq*, *beq*, *Aineq*, *bineq*.
- Los límites inferior y superior de las variables de diseño *lb*, *ub*.
- Opciones varias, almacenadas en *opts*.

En el código desarrollado, la estructura de *createOptimProblem* se almacena en la variable *problem*.

En resumen: se definen los puntos de inicio del proceso de optimización (*GlobalSearch*), se crea la estructura del proceso de optimización (createOptimProblem) con unas opciones determinadas (*optimset*), y éste se inicia con la instrucción apropiada: *run(gs,problem)*.

3. DISEÑO

En este capítulo se proporciona una explicación detallada de los módulos más importantes del programa (en lo referente al propósito de este proyecto) y del funcionamiento de éste en su conjunto.

3.1. FUNCIONAMIENTO DEL PROGRAMA

El flujo de la información es el siguiente:

1- El usuario define los parámetros del problema que va a analizar en su módulo *`ProblemData.m`*.
2- El usuario ejecuta en Matlab el módulo *`GlobalOptimization.m`* y selecciona la carpeta del problema que se va a resolver.
3- El programa se nutre de la información del módulo *`ProblemData.m`*, crea la estructura del proceso de optimización y lo inicia.
4- El proceso de optimización recurre a los módulos *`ObjectiveFunction.m`* y *`RestrictionFunction.m`* para hallar los valores del volumen, las restricciones y las derivadas correspondientes para cada geometría analizada, ya sea mediante la repetición de datos almacenados, la utilización de redes neuronales (con datos cuyo origen está definido en el módulo *`CreateNNetDataPool.m`*) o la realización de análisis MEF.
5- Los datos calculados para cada geometría se almacenan en variables y archivos *.mat*, y los más significativos se escriben en un archivo de texto para dar información del proceso de optimización al usuario.
6- Una vez finalizado el proceso de optimización, si la solución global ha sido hallada mediante cálculos con redes neuronales, se realiza un análisis MEF para comparar resultados e interpretar la validez de los cálculos.
7- Se proporcionan al usuario datos generales útiles de la ejecución del problema en un archivo de texto y se almacenan todos los datos del problema en archivos *.mat*.

A continuación se presenta el diagrama de bloques del proceso descrito:

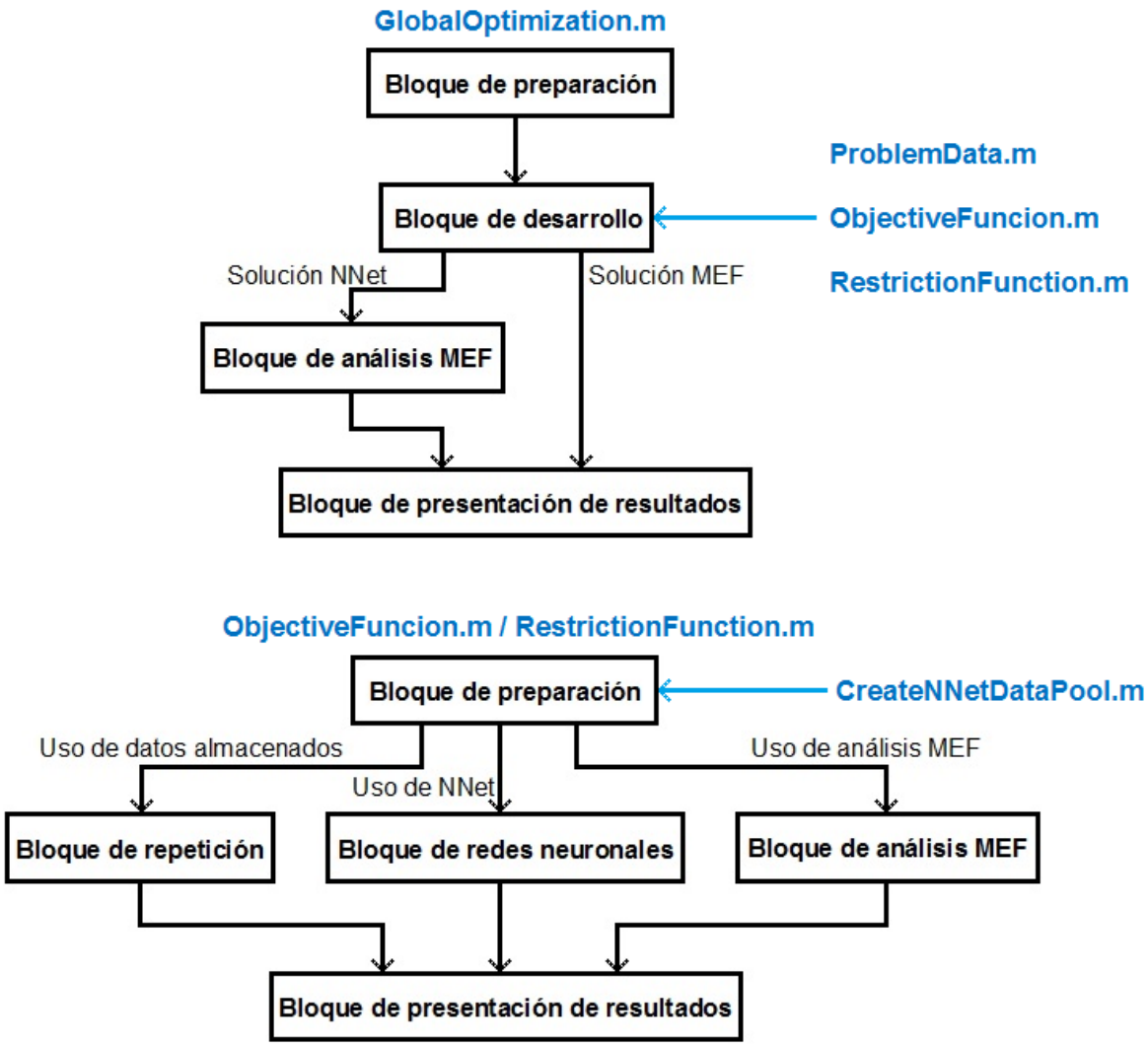

Figura 11. Diagrama de bloques del proceso completo.

3.2. CONTENIDO DE LOS MÓDULOS DEL PROGRAMA

El programa de optimización de forma 2D consta de tres módulos principales y una gran cantidad de módulos “secundarios” necesarios para llevar a cabo, por ejemplo, el análisis de elementos finitos (MEF) de las geometrías.

Los módulos principales, que se van a exponer a continuación, son: *GlobalOptimization.m*, *ObjectiveFunction.m* y *RestrictionFunction.m*.

En este apartado también se explica el contenido de los módulos "secundarios" *CreateNNetDataPool.m* y *ProblemData.m*.

Para conocer más detalles del programa se recomienda la lectura de los manuales del usuario y del programador, que se pueden encontrar en los anexos del proyecto.

3.2.1. Módulo *GlobalOptimization.m*

Es el módulo "maestro" del programa; en él se encuentran las instrucciones que crean las carpetas necesarias para organizar la información de la ejecución del problema, se establecen los parámetros que gobiernan el proceso de optimización, y se inicia la ejecución del mismo, finalizada la cual se presenta al usuario la información pertinente y se almacenan los datos del problema.

Está presente en el directorio principal del programa.

Se presenta un diagrama de bloques que refleja el proceso que se lleva a cabo en este módulo, desarrollado a continuación.

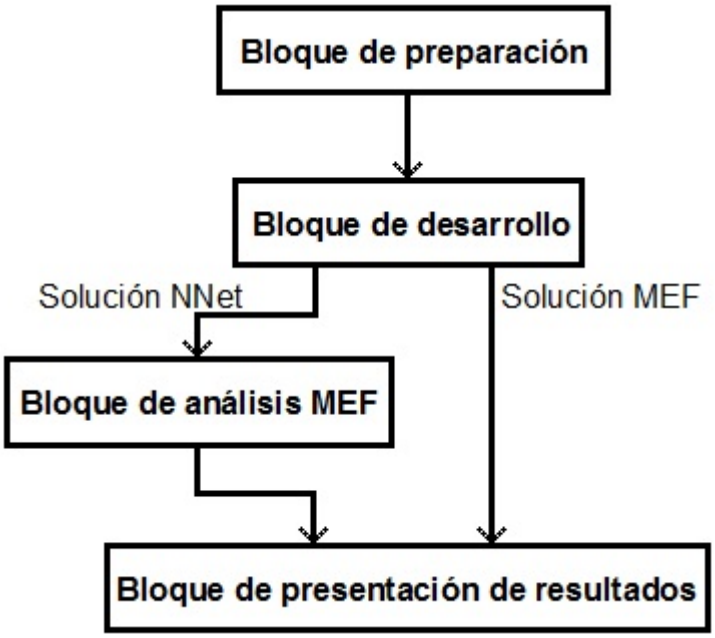

Figura 12. Diagrama de bloques del módulo *GlobalOptimization.m*.

En el “bloque de preparación” se realizan las siguientes instrucciones:

- Se declaran las variables globales necesarias para el funcionamiento del programa.
- Se añaden los directorios que contienen todos los módulos “secundarios” del programa al *path* de Matlab, para que éste pueda utilizarlos.
- El usuario selecciona el problema que quiere resolver.
- Se crean las carpetas donde se almacenarán los datos de la ejecución del problema.

En el “bloque de desarrollo”:

- Se cargan los parámetros definidos por el usuario para el problema en cuestión.
- Se inicializan algunas variables necesarias para el funcionamiento del programa.
- Se crea la estructura del proceso de optimización.
- Se inicia el proceso de optimización.

Una vez que el proceso de optimización ha terminado, la solución global puede haber sido calculada mediante un análisis de elementos finitos, o mediante redes neuronales.

Los análisis MEF son siempre más fiables por lo que, en el segundo escenario, se realiza un análisis MEF para comprobar los resultados hallados mediante redes neuronales.

En el “bloque de análisis MEF” se realiza un análisis de elementos finitos. La explicación detallada se desarrollará en el módulo *`ObjectiveFunction.m`*.

En el “bloque de presentación de resultados”:

- Se procesan y escriben en un archivo de texto los siguientes parámetros:
  - Tiempo total de iteración (tiempo real que ha tardado la ejecución del problema). En este proyecto se considera que cada evaluación de la función objetivo, y cada evaluación de las restricciones, es una iteración del proceso.
  - Tiempo total de análisis (suma de los tiempos de análisis de cada iteración, aunque se hayan usado datos almacenados).
  - Número de iteraciones realizadas.
  - Número y procedencia de los análisis MEF realizados (cuántos se han realizado en el módulo *`ObjectiveFunction.m`* y cuántos en el módulo *`RestrictionFunction.m`*).
  - Número y procedencia de los cálculos con redes neuronales realizados.
  - Número y procedencia de las repeticiones de datos almacenados.
  - Número de la iteración en la que se ha encontrado la solución del problema.
  - Valores solución de las variables de diseño.
  - Valor solución del volumen (función objetivo a minimizar).
  - Error en norma energética en la malla de la iteración solución.
  - Máximo error en norma energética entre todas las iteraciones, e iteración en la que se encuentra.

  Si se comparan datos hallados mediante análisis MEF y redes neuronales, los últimos puntos cambian ligeramente:

  - Valor solución del volumen hallado mediante NNet (redes neuronales).
  - Valor solución del volumen hallado mediante análisis MEF.
  - Valores de las restricciones hallados mediante análisis MEF.
  - Error en norma energética hallado mediante análisis MEF.

- Se almacenan los datos del problema en archivos *.mat*.

3.2.2. Módulo *ObjectiveFunction.m*

Es uno de los dos módulos que componen el proceso de optimización. Se encarga de calcular el valor de la función objetivo a minimizar (volumen) y sus derivadas para unos valores determinados de las variables de diseño.

Está presente en el directorio principal del programa.

Se presenta un diagrama de bloques que refleja el proceso que se lleva a cabo en este módulo, desarrollado a continuación.

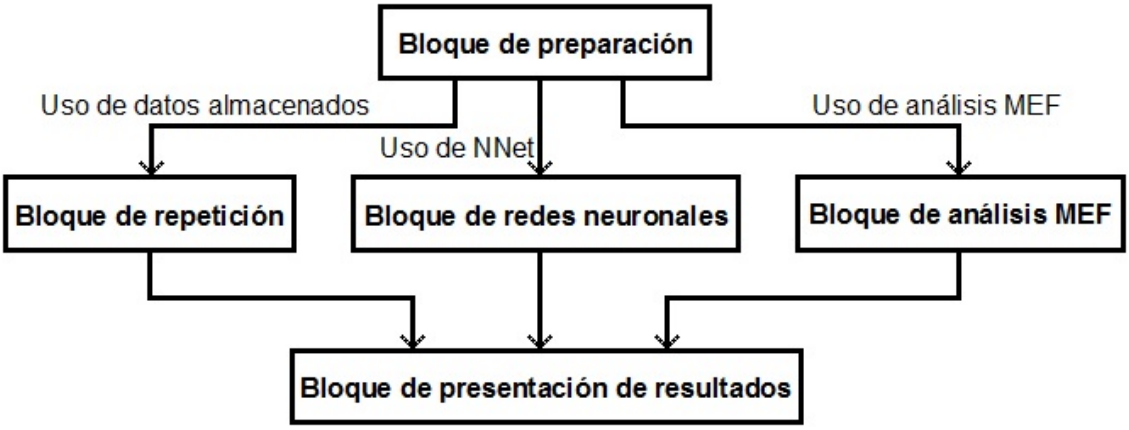

Figura 13. Diagrama de bloques del módulo *ObjectiveFunction.m*.

En el “bloque de preparación”:

- Se declaran las variables globales necesarias para el funcionamiento del programa.
- Se construye el conjunto de datos que el programa tiene en cuenta a la hora de detectar la repetición de los valores de las variables de diseño sugeridos por el programa de optimización.
- El programa comprueba si se produce dicha repetición.
- Si no se produce la repetición de datos y el usuario ha decidido utilizar redes neuronales para hacer cálculos, se construye el conjunto de datos que el programa tiene en cuenta a la hora de construir las redes neuronales. Se creará

una red neuronal para calcular el volumen y sus derivadas, y otra por cada restricción del proceso de optimización. Si existe un número de datos útiles suficiente, el programa procede a construirlas en el siguiente bloque.

- Si, por el contrario, el programa ya está trabajando con unas redes neuronales determinadas y el proceso de optimización sugiere unos valores de las variables de diseño que están fuera del rango de datos usados para entrenar dichas redes neuronales, se procede al análisis MEF en el siguiente bloque y las redes neuronales existentes son descartadas.

En el “bloque de repetición”, al cual accede el programa cuando se detecta la repetición de los valores de las variables de diseño sugeridos por el programa de optimización:

- Se pone a disposición inmediata del programa toda la información necesaria para dar salida al módulo *`ObjectiveFuncion.m`* (el volumen y sus derivadas).

En el “bloque de redes neuronales”:

- Si ya están definidas unas redes neuronales determinadas, se utilizan para obtener los valores del volumen y sus derivadas y los de las restricciones y sus derivadas.
- Si no hay unas redes neuronales definidas, se procesa la información disponible para construirlas. Se crea una red neuronal para calcular el volumen y sus derivadas, y otra por cada restricción del proceso de optimización, y se utilizan para obtener los valores mencionados.

En el “bloque de análisis MEF”:

- Se inicializan determinadas variables necesarias para realizar el análisis MEF y se definen los parámetros del proceso de mallado según los valores designados por el usuario en el módulo *`ProblemData.m`*.
- Se ejecuta el calculador que realiza el análisis MEF.
- Se calculan los valores del volumen y sus derivadas.
- Se calculan los valores de las restricciones y sus derivadas.

Si se produce algún error en el análisis MEF, cambia el proceso:

- Se crean las carpetas necesarias para almacenar los datos y la geometría del error ocurrido.

- Se dibuja y almacena la geometría que da lugar al error y se escribe en un documento de texto toda la información de dicho error.
- Al volumen y sus derivadas se les asignan valores NaN (Not a Number), con los que el proceso de optimización puede trabajar.
- A las restricciones se les asignan deliberadamente valores que excluyen a la geometría que produce el error de ser considerada como solución por el proceso de optimización.

En el "bloque de presentación de resultados":

- Si la geometría propuesta por el proceso de optimización es aceptable (cumple todas las restricciones) y los datos calculados provienen de un análisis MEF, se dibuja y almacena dicha geometría.
- Se escriben en un archivo de texto los siguientes datos:
  - Número de iteración.
  - Origen de la iteración (repetición de datos/ redes neuronales/ análisis MEF).
  - Valores de las variables de diseño.
  - Valor del volumen.
  - Valores de las restricciones.
  - Error en norma energética.
  - Tiempo de iteración.
- Se guardan los datos de la iteración en un archivo *.mat*.

3.2.3. Módulo *RestrictionFunction.m*

Es uno de los dos módulos que componen el proceso de optimización. Se encarga de calcular el valor de las restricciones y sus derivadas para unos valores determinados de las variables de diseño.

Está presente en el directorio principal del programa.

El proceso que se lleva a cabo en este módulo es idéntico al del módulo `ObjectiveFunction.m` con la única diferencia de que, en esta ocasión, las salidas son las restricciones y sus derivadas.

3.2.4. Módulo *CreateNNetDataPool.m*

En este módulo se define el contenido del conjunto de datos que el programa tiene en cuenta a la hora de construir las redes neuronales.

Está presente en el directorio principal del programa.

En el momento de la redacción de este proyecto, sólo hay implementada una estrategia: la utilización conjunta de todos los datos almacenados en las variables `ResultData` (en ella se almacenan por columnas datos significativos de cada iteración del proceso de optimización) y `PrevResData` (que es la variable `ResultData` de una ejecución previa del problema).

3.2.5. Módulo *ProblemData.m*

En este módulo el usuario define los parámetros de una ejecución determinada del programa.

Está presente en el directorio de cada problema.

Los parámetros definidos son:

- Valores iniciales de las variables de diseño, restricciones de igualdad y desigualdad entre ellas, y límites inferior y superior de dichos valores.
- Parámetros que definen la estructura y las propiedades del proceso de optimización.
- El uso de datos almacenados de ejecuciones anteriores del mismo problema, así como la utilización de redes neuronales en la realización de cálculos.
- Estrategia a la hora de definir el contenido del conjunto de datos que el programa tiene en cuenta a la hora de construir las redes neuronales y varios parámetros necesarios para dicha construcción.
- Valores que definen el origen de los datos y las iteraciones.
- Número de restricciones que tiene en cuenta el proceso de optimización para alcanzar la solución.

- Ejes fijos que se usarán para representar las geometrías correspondientes.
- Inicialización de variables relacionadas con los análisis MEF.
- Geometría del problema que se va a analizar y propiedades del material.
- Asignación de las coordenadas de determinados puntos de la geometría a las variables de diseño correspondientes y establecimiento de la función encargada de actualizar la geometría con los valores de las variables de diseño.
- Parámetros necesarios para el proceso de mallado del bloque de análisis MEF.

Aquí finaliza la exposición de los contenidos de los módulos más importantes del programa (en lo referente al propósito de este proyecto). Para más información, se recomienda la lectura de los manuales del usuario y del programador, que se pueden encontrar en los anexos del proyecto.

4. RESULTADOS

En los problemas que el programa tiene que resolver, se busca la minimización del volumen de material utilizado en el componente a optimizar considerando la restricción de la tensión máxima de Von Mises. Las variables de diseño son las coordenadas de determinados puntos de la geometría que se analiza.

En este capítulo se muestran los resultados de distintas ejecuciones de varios problemas realizadas con el programa desarrollado en este proyecto con el fin de comprobar que se ha cumplido el objetivo de reducir el tiempo de ejecución del programa anterior.

En dicho programa, el tiempo de ejecución de un problema determinado dependía únicamente de:

- Número de puntos de inicio potenciales a examinar.
- Tiempo necesario para cada una de las iteraciones del proceso de mallado h-adaptativo de los análisis de elementos finitos.

Se han introducido dos mejoras para reducirlo: la utilización de resultados almacenados en la ejecución en curso y en anteriores ejecuciones, y la realización de cálculos con redes neuronales (*redes*, en plural, pues se construye una red neuronal para el conjunto volumen-gradiente del volumen, y otra por cada conjunto restricción-gradiente de la restricción que tenga el problema).

Con la implementación de la reutilización de resultados almacenados, el tiempo de ejecución usando el nuevo programa depende de un factor adicional, además de los dos ya mencionados: la variedad en los valores sugeridos de las variables de diseño por el proceso de optimización. Cuando el usuario define el número de puntos de inicio potenciales a examinar en el proceso de optimización, cabe la posibilidad de que varios de esos puntos conduzcan a dicho proceso a sugerir unos valores idénticos de las variables de diseño, y por tanto a analizar geometrías idénticas.

Con el uso de resultados almacenados en la misma ejecución del problema y en ejecuciones anteriores, es posible disponer al instante de los resultados de los análisis que se llevarían a cabo con esas geometrías, permitiendo ahorrar el tiempo que llevaría

realizar dichos análisis de nuevo a la hora de evaluar tanto la función objetivo como las restricciones.

Con la implementación de la utilización de redes neuronales (NNet) para hacer cálculos, el tiempo de ejecución de un problema determinado usando el nuevo programa depende de varios factores adicionales, además de los ya mencionados:

- El número mínimo de datos provenientes de análisis MEF diferentes que se requiere para la creación de las redes neuronales, obtenidos en la misma ejecución del problema o en anteriores.
- El tiempo de entrenamiento de las redes neuronales, y el número de ellas que se creen en bucle para la selección de las que tengan menor error de validación.
- El número máximo de usos consecutivos de unas redes neuronales determinadas (siempre será más rápido un cálculo realizado con redes neuronales que otro realizado con un análisis MEF).
- El número máximo de análisis MEF entre el fin de la utilización de un conjunto de redes y la construcción del siguiente, para que tenga más datos de entrada con los que trabajar (lo que se traduce en tiempo en el que no se estan usando redes neuronales para realizar cálculos).
- El hecho de que el programa de optimización podría sugerir unos valores de las variables de diseño que estén fuera del rango de entrenamiento de las redes neuronales en uso, forzando el descarte de éstas, aunque hayan sido utilizadas una única vez, y se tenga que repetir todo el proceso de creación desde el principio.

Como se puede apreciar, la introducción de redes neuronales abre todo un mundo de posibilidades para la disminución del tiempo de ejecución del programa, y no todas están bajo el control del usuario. La estrategia a seguir se debe determinar en cada problema, pudiendo ser más o menos agresiva.

Este proyecto se ha realizado considerando que cada vez que el programa evalúa la función objetivo o las restricciones de un problema, realiza una iteración. Esto conduce a la consideración de dos tiempos: el “tiempo de iteración” y el “tiempo de análisis” para comparar los efectos de las mejoras implementadas. Se sigue esta estrategia: en una determinada iteración del proceso, si la geometría propuesta no es repetida, es analizada mediante MEF o redes neuronales; en ese caso, el tiempo de la iteración coincide con el

tiempo del análisis MEF o de redes neuronales. Si, por el contrario, la geometría propuesta es repetida, el programa obtiene de inmediato los resultados del análisis de esa geometría, pues están almacenados; en ese caso, el tiempo de análisis es el que necesitó el programa en su momento para realizar el análisis MEF o de redes neuronales, y el tiempo de iteración es un instante (lo que tarda el programa en leer los resultados almacenados).

Sumando todos los tiempos de iteración se obtiene el “tiempo de iteración del problema”, y sumando todos los tiempos de análisis se obtiene el “tiempo de análisis del problema”. El “tiempo de iteración del problema” es, pues, el tiempo que transcurre desde el comienzo de la ejecución del problema hasta la finalización de ésta, es decir, el tiempo *real* de la ejecución del problema. El “tiempo de análisis del problema” coincidiría con el “tiempo de iteración del problema” si el programa no utilizara resultados almacenados en el proceso de optimización y tuviera que realizar análisis MEF o de redes neuronales en cada iteración del proceso.

Para cada problema:

- Se analiza primero el efecto del uso de resultados almacenados, y después el de ambas mejoras a la vez.
- Para cada efecto estudiado, se realizan tres análisis:
  - Un primer análisis con un número de puntos de inicio determinados.
  - Un segundo análisis con el mismo número de puntos de inicio que el primero utilizando los resultados almacenados en éste (simulando la continuación de un análisis tras una parada).
  - Un tercer análisis con el doble de puntos de inicio que el primero utilizando los resultados almacenados en éste (buscando el mínimo global desde un mayor número de puntos de inicio).

Realizando todos esos análisis se pretende estudiar:

- El ahorro de tiempo que suponen las mejoras desde la primera ejecución del problema.
- La variedad en los valores de las variables de diseño sugeridos por el proceso de optimización.
- La cantidad de mínimos locales que tiene el problema.

4.1. PROBLEMA CILINDRO 1

Este problema consiste en una tubería con superficie interna circular sometida a presión interna y radio externo a optimizar.

Sólo se considera una variable de diseño, que representa al radio.

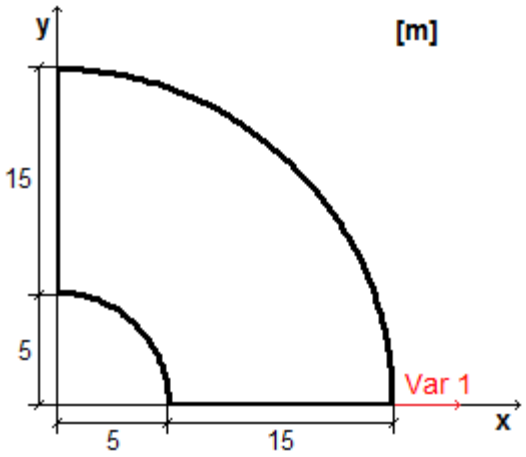

Figura 14. Geometría inicial y variable de diseño del problema Cilindro 1.

4.1.1. Uso de resultados almacenados

En este subapartado se estudia el efecto de la mejora que permite la utilización de resultados almacenados en la ejecución en curso y en anteriores ejecuciones del problema.

Primera ejecución del problema.

La primera ejecución de este problema (esto es, sin disponibilidad alguna de datos almacenados de ejecuciones anteriores) se realiza con:

- 10 puntos de inicio potenciales.
- 1 restricción (tensión máxima admisible de Von Mises $2*10^6$ Pa).
- Elementos cuadráticos.
- Error objetivo en norma energética: 1%.
- Número máximo de iteraciones del proceso de mallado h-adaptativo: 3.
- Tolerancias del criterio de parada (tanto *TolX* como *TolFun*): 10^{-3}. Son tolerancias absolutas de las que se valdrá el programa para saber que ha llegado a la solución. Si la norma de $(X_i - X_{i+1})$ es menor que *TolX* y la norma de $(f(X_i) - f(X_{i+1}))$ es menor que *TolFun*, el programa considera que se ha llegado a la solución. *TolX* es 10^{-3} m y *TolFun* es 10^{-3} m^3.

Se llega a la siguiente solución:

- Valor de la variable de diseño: 10’6944 m.
- Volumen: 70’3964 m^3.

Con:

- 259 geometrías consideradas en el proceso.
- 74 geometrías analizadas con MEF.
- 185 geometrías reutilizadas.
- Tiempo promedio de análisis MEF: 13’4166 seg.

El tiempo de análisis del problema es: 57 min 54 seg.

El tiempo de iteración del problema es: 14 min 19 seg.

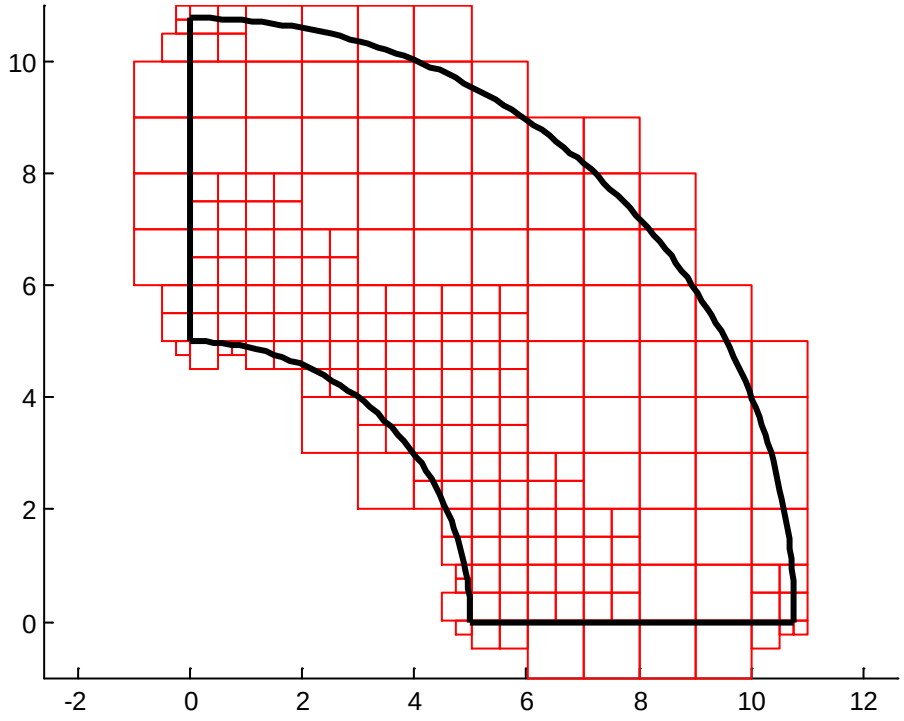

Figura 15. Geometría solución del problema Cilindro 1.

Segunda ejecución del problema.

La segunda ejecución del problema se realiza con los mismos parámetros que la primera, pero con la disposición de los datos almacenados en ésta.

Se llega a la siguiente solución:

- Valor de la variable de diseño: 10’6944 m.
- Volumen: 70’3964 m^3.

Con:

- 259 geometrías consideradas en el proceso.
- 12 geometrías analizadas con MEF.
- 247 geometrías reutilizadas.

- Tiempo promedio de análisis MEF: 12’9859 seg.

El tiempo de análisis del problema es: 56 min 3 seg.

El tiempo de iteración del problema es: 2 min 54 seg.

La geometría es idéntica a la de la primera ejecución.

Tercera ejecución del problema.

La tercera ejecución del problema se realiza con los mismos parámetros que la primera salvo el número de puntos de inicio, que es el doble (20), y con la disposición de los datos almacenados en ésta.

Se llega a la siguiente solución:

- Valor de la variable de diseño: 10’6944 m.
- Volumen: 70’3964 m^3.

Con:

- 287 geometrías consideradas en el proceso.
- 11 geometrías analizadas con MEF.
- 276 geometrías reutilizadas.
- Tiempo promedio de análisis MEF: 13’7887 seg.

El tiempo de análisis del problema es: 1 h 5 min 57 seg.

El tiempo de iteración del problema es: 2 min 52 seg.

La geometría es idéntica a la de la primera ejecución.

Recopilando los tiempos obtenidos:

Tabla 1. Comparación de tiempos del problema Cilindro 1.

Cilindro 1	**1ª ejecución 10 puntos**	**2ª ejecución 10 puntos**	**3ª ejecución 20 puntos**
Tiempo de análisis	57 min 54 seg	56 min 3 seg	1 h 5 min 57 seg
Tiempo de iteración	14 min 19 seg	2 min 54 seg	2 min 52 seg
Ahorro de tiempo	43 min 35 seg	53 min 9 seg	1 h 3 min 5 seg
Ahorro de tiempo (porcentaje)	75'27%	94'83%	95'65%

Las conclusiones que se sacan de los resultados son:

- El uso de datos almacenados supone un ahorro significativo de tiempo desde la primera ejecución del problema.
- Los reducidos tiempos de iteración de los dos últimos análisis evidencian un gran número de usos de datos almacenados; no hay mucha variedad en los valores sugeridos de la variable de diseño por el proceso de optimización.
- La similaridad entre los tiempos de análisis de las ejecuciones de 10 puntos de inicio y la de 20 demuestra que el problema tiene relativamente pocos mínimos locales entre los que buscar el global; de hecho, este problema solamente tiene uno.

4.1.2. Uso de resultados almacenados y redes neuronales

En este subapartado se estudia el efecto de utilizar ambas mejoras propuestas al mismo tiempo.

Dada la sencillez de este problema, se plantea una estrategia agresiva para demostrar el potencial de las redes neuronales: realizar una primera ejecución con un único punto de

inicio donde sólo se realizan análisis MEF, y una segunda ejecución con 20 puntos de inicio potenciales, que dispone de los resultados calculados en la primera, donde sólo se emplean redes neuronales para realizar cálculos.

Primera ejecución del problema.

La primera ejecución de este problema (esto es, sin disponibilidad alguna de datos almacenados de ejecuciones anteriores) se realiza con los mismos parámetros expuestos en el subapartado anterior, pero con un único punto de inicio.

Se llega a la siguiente solución:

- Valor de la variable de diseño: 10’6944 m.
- Volumen: 70’3964 m^3.

Con:

- 241 geometrías consideradas en el proceso.
- 74 geometrías analizadas con MEF.
- 167 geometrías reutilizadas.
- Tiempo promedio de análisis MEF: 12’5284 seg.

El tiempo de análisis del problema es: 50 min 19 seg.

El tiempo de iteración del problema es: 13 min 44 seg.

La geometría es idéntica a la del subapartado anterior.

Segunda ejecución del problema.

La segunda ejecución del problema se realiza con los mismos parámetros que la primera salvo el número de puntos de inicio (ahora son 20), con la disposición de los datos almacenados en ésta, y con los siguientes parámetros relacionados con las redes neuronales:

- 74 datos MEF como mínimo para la construcción de las redes neuronales (todos los análisis MEF realizados en la primera ejecución).
- 3 redes neuronales para cada conjunto de datos (volumen-gradiente del volumen, restricción-gradiente de la restricción) se construyen en un bucle para la elección de las que tengan el menor error de validación.
- 50 usos consecutivos de las redes neuronales para hacer cálculos (se pretende utilizar sólo redes neuronales en esta ejecución, por eso el número de usos es tan elevado).

Se llega a la siguiente solución:

- Valor de la variable de diseño: 10’6944 m.
- Volumen: 70’3964 m^3.

Con:

- 287 geometrías consideradas en el proceso.
- 0 geometrías analizadas con MEF.
- 14 geometrías analizadas con redes neuronales.
- 273 geometrías reutilizadas.

El tiempo de análisis del problema es: 44 min 28 seg.

El tiempo de iteración del problema es: 20 seg.

La geometría es idéntica a la de la primera ejecución.

Recopilando los tiempos obtenidos:

Tabla 2. Comparación de tiempos del problema Cilindro 1 con estrategia NNet.

Cilindro 1	**1ª ejecución** **1 punto sólo MEF**	**2ª ejecución** **20 puntos sólo NNet**
Tiempo de análisis	50 min 19 seg	44 min 28 seg
Tiempo de iteración	13 min 44 seg	20 seg
Ahorro de tiempo	36 min 35 seg	44 min 8 seg
Ahorro de tiempo (porcentaje)	72’71%	99’25%

Las conclusiones que se sacan de los resultados son:

- El uso de datos almacenados supone un ahorro significativo de tiempo en la primera ejecución del problema.
- El uso de datos almacenados, redes neuronales y la estrategia explicada supone un ahorro sorprendente de tiempo en la segunda ejecución del problema.
- El reducido tiempo de iteración del segundo análisis evidencia un gran número de usos de datos almacenados; no hay mucha variedad en los valores sugeridos de la variable de diseño por el proceso de optimización.
- La poca diferencia entre los tiempos de análisis de las ejecuciones demuestra que el problema tiene relativamente pocos mínimos locales entre los que buscar el global; de hecho, este problema solamente tiene uno.
- Al utilizarse sólo redes neuronales en la segunda ejecución, el tiempo de iteración depende casi exclusivamente del tiempo de entrenamiento de éstas.
- La diferencia entre los tiempos de análisis se debe a la procedencia de los datos. En el primer análisis se realizan 74 análisis MEF y 167 repeticiones de datos almacenados; en el segundo se realizan 14 cálculos NNet y 273 repeticiones de datos almacenados.

  Se deduce que los datos calculados con redes neuronales se han repetido muchas veces en la segunda ejecución, de ahí que el tiempo de análisis de ésta sea inferior al de la primera.

La utilización de esta agresiva estrategia no se emplea en el resto de problemas, pues son mucho más complejos; requieren la utilización conjunta de análisis MEF y redes neuronales en todas las ejecuciones.

4.2. PROBLEMA CILINDRO 2

Este problema consiste en una tubería con paredes rectas y arco en superficie interna sometida a presión interna y radio externo a optimizar.

Se consideran cuatro variables de diseño.

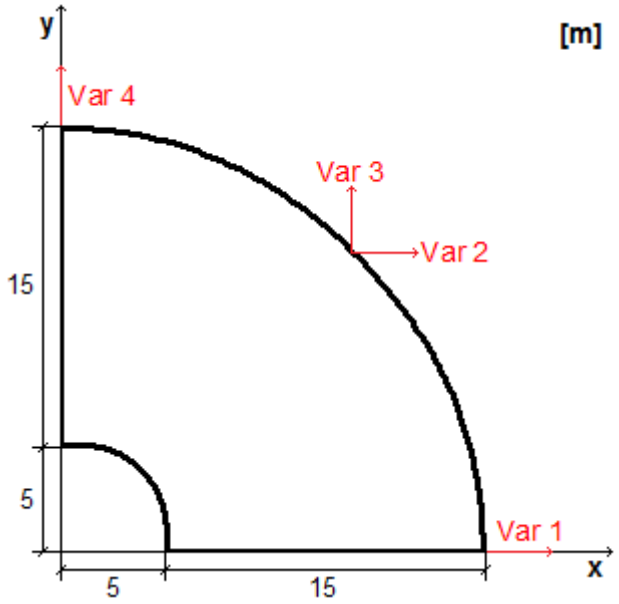

Figura 16. Geometría inicial y variables de diseño del problema Cilindro 2.

4.2.1. Uso de resultados almacenados

En este subapartado se estudia el efecto de la mejora que permite la utilización de resultados almacenados en la ejecución en curso y en anteriores ejecuciones del problema.

Primera ejecución del problema.

La primera ejecución de este problema (esto es, sin disponibilidad alguna de datos almacenados de ejecuciones anteriores) se realiza con:

- 20 puntos de inicio potenciales.
- 1 restricción (tensión máxima admisible de Von Mises $2*10^6$ Pa).
- Elementos cuadráticos.
- Error objetivo en norma energética: 1%.
- Número máximo de iteraciones del proceso de mallado h-adaptativo: 3.
- Tolerancias del criterio de parada (tanto *TolX* como *TolFun*): 10^{-3}. Son tolerancias absolutas de las que se valdrá el programa para saber que ha llegado a la solución. Si la norma de $(X_i - X_{i+1})$ es menor que *TolX* y la norma de $(f(X_i) - f(X_{i+1}))$ es menor que *TolFun*, el programa considera que se ha llegado a la solución. *TolX* es 10^{-3} m y *TolFun* es 10^{-3} m^3.

Se llega a la siguiente solución:

- Valor de las variables de diseño: [17'5758, 10'4929, 10'443, 17'6255] m.
- Volumen: 185'3 m^3.

Con:

- 239 geometrías consideradas en el proceso.
- 52 geometrías analizadas con MEF.
- 187 geometrías reutilizadas.
- Tiempo promedio de análisis MEF: 132'5817 seg.

El tiempo de análisis del problema es: 8 h 48 min 7 seg.

El tiempo de iteración del problema es: 1 h 51 min 7 seg.

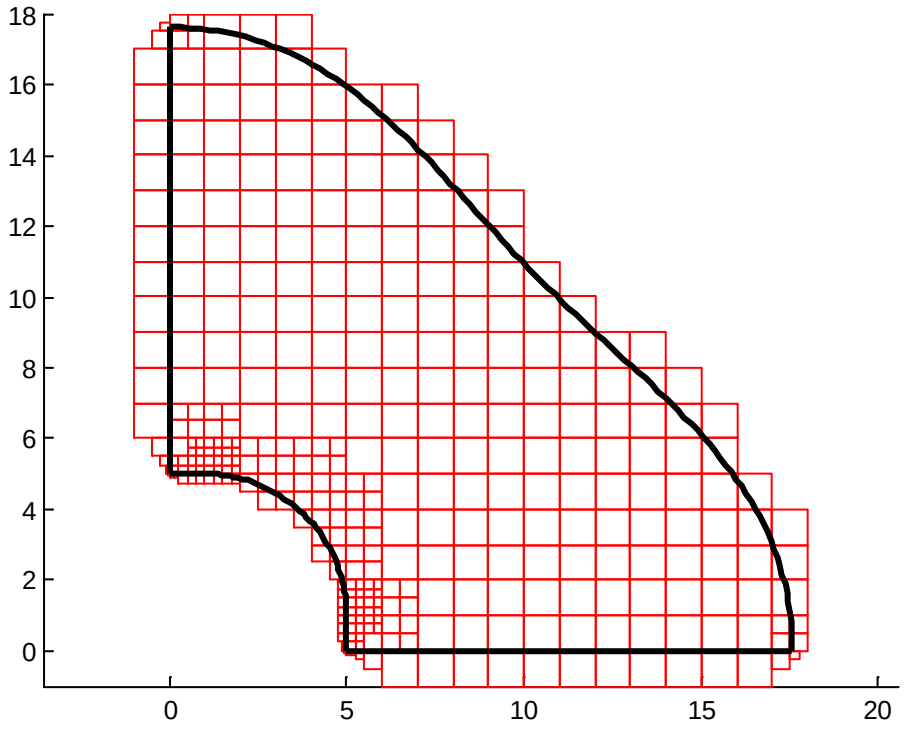

Figura 17. Geometría solución del problema Cilindro 2.

Segunda ejecución del problema.

Se propone una segunda ejecución del problema con los mismos parámetros que la primera, pero con la disposición de los datos almacenados en ésta.

Se llega a la siguiente solución:

- Valor de las variables de diseño: [17’5758, 10’4929, 10’443, 17’6255] m.
- Volumen: 185’3 m^3.

Con:

- 239 geometrías consideradas en el proceso.
- 16 geometrías analizadas con MEF.
- 223 geometrías reutilizadas.
- Tiempo promedio de análisis MEF: 135’2897 seg.

El tiempo de análisis del problema es: 8 h 58 min 54 seg.

El tiempo de iteración del problema es: 41 min 51 seg.

La geometría es idéntica a la de la primera ejecución.

Tercera ejecución del problema.

La tercera ejecución del problema se realiza con los mismos parámetros que la primera salvo el número de puntos de inicio, que es el doble (40), y con la disposición de los datos almacenados en ésta.

Se llega a la siguiente solución:

- Valor de las variables de diseño: [17’5758, 10’4929, 10’443, 17’6255] m.
- Volumen: 185’3 m^3.

Con:

- 319 geometrías consideradas en el proceso.
- 36 geometrías analizadas con MEF.
- 283 geometrías reutilizadas.
- Tiempo promedio de análisis MEF: 140’6946 seg.

El tiempo de análisis del problema es: 12 h 28 min 1 seg.

El tiempo de iteración del problema es: 1 h 34 min 44 seg.

La geometría es idéntica a la de la primera ejecución.

Recopilando los tiempos obtenidos:

Tabla 3. Comparación de tiempos del problema Cilindro 2.

Cilindro 2	**1ª ejecución 20 puntos**	**2ª ejecución 20 puntos**	**3ª ejecución 40 puntos**
Tiempo de análisis	8 h 48 min 7 seg	8 h 58 min 54 seg	12 h 28 min 1 seg
Tiempo de iteración	1 h 51 min 7 seg	41 min 51 seg	1 h 34 min 44 seg
Ahorro de tiempo	6 h 57 min	8 h 17 min 3 seg	10 h 53 min 17 seg
Ahorro de tiempo (porcentaje)	78’96%	92’23%	87’34%

Las conclusiones que se sacan de los resultados son:

- El uso de datos almacenados supone un ahorro significativo de tiempo desde la primera ejecución del problema.
- Los tiempos de iteración de los dos últimos análisis evidencian un número no muy elevado de usos de datos almacenados; hay mucha variedad en los valores sugeridos de las variables de diseño por el proceso de optimización.
- La gran diferencia entre los tiempos de análisis de las ejecuciones de 20 puntos de inicio y la de 40 demuestra que el problema tiene muchos mínimos locales entre los que buscar el global.

4.2.2. Uso de resultados almacenados y redes neuronales

En este subapartado se estudia el efecto de utilizar ambas mejoras propuestas al mismo tiempo.

Primera ejecución del problema.

La primera ejecución del problema (esto es, sin disponibilidad alguna de datos almacenados de ejecuciones anteriores) se realiza con los mismos parámetros de la primera ejecución del subapartado anterior, y con los siguientes parámetros relacionados con las redes neuronales:

- 30 datos MEF como mínimo para la construcción de las redes neuronales.
- 4 redes neuronales para cada conjunto de datos (volumen-gradiente del volumen, restricción-gradiente de la restricción) se construyen en un bucle para la elección de las que tengan el menor error de validación.
- 5 usos consecutivos de las redes neuronales para hacer cálculos.
- 5 análisis MEF se realizan entre la utilización de un conjunto de redes neuronales y el siguiente.

Se llega a la siguiente solución:

- Valor de las variables de diseño: [17’6157, 10’4558, 10’456, 17’6237] m.
- Volumen: 185’308 m^3.

Con:

- 186 geometrías consideradas en el proceso.
- 35 geometrías analizadas con MEF.
- 6 geometrías analizadas con redes neuronales.
- 145 geometrías reutilizadas.

El tiempo de análisis del problema es: 6 h 17 min 8 seg.

El tiempo de iteración del problema es: 1 h 18 min 29 seg.

La geometría es prácticamente idéntica a la del subapartado anterior; el programa se ha decantado por un mínimo muy próximo al anterior.

Segunda ejecución del problema.

La segunda ejecución del problema se realiza con los mismos parámetros que la primera, pero con la disposición de los datos almacenados en ésta.

Se llega a la siguiente solución:

- Valor de las variables de diseño: [17’6157, 10’4558, 10’456, 17’6237] m.
- Volumen: 185’308 m^3.

Con:

- 190 geometrías consideradas en el proceso.
- 10 geometrías analizadas con MEF.
- 11 geometrías analizadas con redes neuronales.
- 169 geometrías reutilizadas.

El tiempo de análisis del problema es: 4 h 48 min 40 seg.

El tiempo de iteración del problema es: 24 min 56 seg.

La geometría es idéntica a la de la primera ejecución.

Tercera ejecución del problema.

La tercera ejecución del problema se realiza con los mismos parámetros que la primera salvo el número de puntos de inicio, que es el doble (40), y con la disposición de los datos almacenados en ésta.

Se llega a la siguiente solución:

- Valor de las variables de diseño: [17’6157, 10’4558, 10’456, 17’6237] m.
- Volumen: 185’308 m^3.

Con:

- 268 geometrías consideradas en el proceso.

- 20 geometrías analizadas con MEF.
- 21 geometrías analizadas con redes neuronales.
- 227 geometrías reutilizadas.

El tiempo de análisis del problema es: 7 h 8 min 57 seg.

El tiempo de iteración del problema es: 54 min 30 seg.

La geometría es idéntica a la de la primera ejecución.

Recopilando los tiempos obtenidos:

Tabla 4. Comparación de tiempos del problema Cilindro 2 con estrategia NNet.

Cilindro 2	**1ª ejecución 20 puntos**	**2ª ejecución 20 puntos**	**3ª ejecución 40 puntos**
Tiempo de análisis	6 h 17 min 8 seg	4 h 48 min 40 seg	7 h 8 min 57 seg
Tiempo de iteración	1 h 18 min 29 seg	24 min 56 seg	54 min 30 seg
Ahorro de tiempo	4 h 58 min 39 seg	4 h 23 min 44 seg	6 h 14 min 27 seg
Ahorro de tiempo (porcentaje)	79’19%	91’36%	87’29%

La utilización de redes neuronales hace más difícil sacar conclusiones precisas de los tiempos de análisis, ya que el azar influye considerablemente (sobre todo en el tiempo de entrenamiento de las redes neuronales, que depende de valores iniciales aleatorios, y en los valores de las variables de diseño sugeridos por el proceso de optimización). Con todo, se puede concluir que:

- El uso de datos almacenados y redes neuronales supone un ahorro significativo de tiempo desde la primera ejecución del problema.
- Los tiempos de iteración de los dos últimos análisis muestran un número no muy elevado de usos de datos almacenados; hay mucha variedad en los valores sugeridos de las variables de diseño por el proceso de optimización.

- La diferencia entre los tiempos de análisis de la 1ª y la 2ª ejecución con 20 puntos de inicio evidencia un uso mayor de cálculos con redes neuronales en la última, ya que ésta dispone desde el principio de los datos de análisis MEF de la ejecución anterior.
- La gran diferencia entre los tiempos de análisis de las ejecuciones de 20 puntos de inicio y la de 40 demuestra que el problema tiene muchos mínimos locales entre los que buscar el global.

A continuación se presenta la comparación de tiempos definitiva del problema: los tiempos de análisis cuando sólo se utiliza la repetición de datos, y los tiempos de iteración cuando se utilizan análisis MEF y redes neuronales. De esta forma se comprueba el ahorro de tiempo total entre el tiempo de ejecución del anterior programa y el del nuevo cuando éste utiliza todo su potencial:

Tabla 5. Comparación de tiempos definitiva del problema Cilindro 2.

Cilindro 2	**1ª ejecución 20 puntos**	**2ª ejecución 20 puntos**	**3ª ejecución 40 puntos**
Tiempo de análisis (sólo MEF)	8 h 48 min 7 seg	8 h 58 min 54 seg	12 h 28 min 1 seg
Tiempo de iteración MEF + NNet	1 h 18 min 29 seg	24 min 56 seg	54 min 30 seg
Ahorro de tiempo total	7 h 29 min 38 seg	8 h 33 min 58 seg	11 h 33 min 31 seg
Ahorro de tiempo total (porcentaje)	85’14%	95’37%	92’71%

El ahorro de tiempo podría haber sido aún mayor si se hubiera empleado una estrategia de redes neuronales más agresiva (por ejemplo, mediante el aumento del número de usos consecutivos de las redes neuronales para hacer cálculos).

4.3. PROBLEMA CILINDRO 3

Este problema consiste en una tubería con superficie interna elíptica sometida a presión interna y radio externo a optimizar.

Se consideran cuatro variables de diseño.

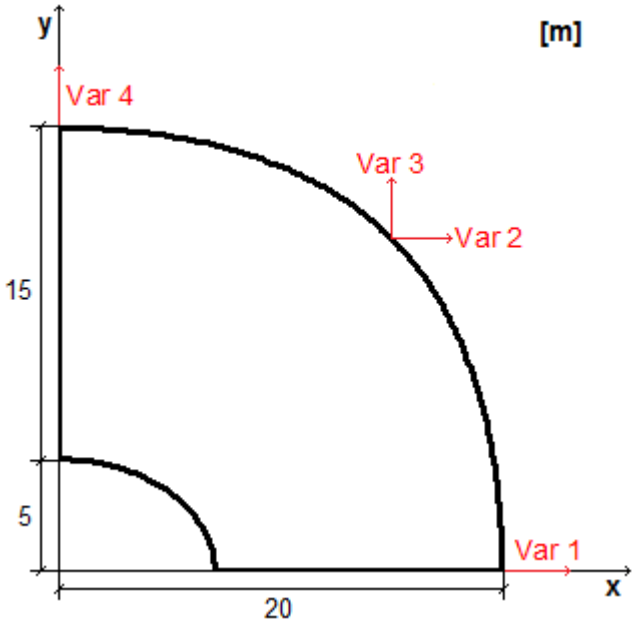

Figura 18. Geometría inicial y variables de diseño del problema Cilindro 3.

4.3.1. Uso de resultados almacenados

En este subapartado se estudia el efecto de la mejora que permite la utilización de resultados almacenados en la ejecución en curso y en anteriores ejecuciones del problema.

Primera ejecución del problema.

La primera ejecución de este problema (esto es, sin disponibilidad alguna de datos almacenados de ejecuciones anteriores) se realiza con:

- 20 puntos de inicio potenciales.
- 1 restricción (tensión máxima admisible de Von Mises 2'7*10^6 Pa).
- Elementos cuadráticos.
- Error objetivo en norma energética: 1%.
- Número máximo de iteraciones del proceso de mallado h-adaptativo: 3.
- Tolerancias del criterio de parada (tanto *TolX* como *TolFun*): 10^{-3}. Son tolerancias absolutas de las que se valdrá el programa para saber que ha llegado a la solución. Si la norma de (X_i-X_{i+1}) es menor que *TolX* y la norma de ($f(X_i)$-$f(X_{i+1})$) es menor que *TolFun*, el programa considera que se ha llegado a la solución. *TolX* es 10^{-3} m y *TolFun* es 10^{-3} m^3.

Se llega a la siguiente solución:

- Valor de las variables de diseño: [18'8048, 13'4874, 13'4874, 18'5099] m.
- Volumen: 252'301 m^3.

Con:

- 301 geometrías consideradas en el proceso.
- 81 geometrías analizadas con MEF.
- 220 geometrías reutilizadas.
- Tiempo promedio de análisis MEF: 116'7511 seg.

El tiempo de análisis del problema es: 9 h 45 min 42 seg.

El tiempo de iteración del problema es: 2 h 20 min 54 seg.

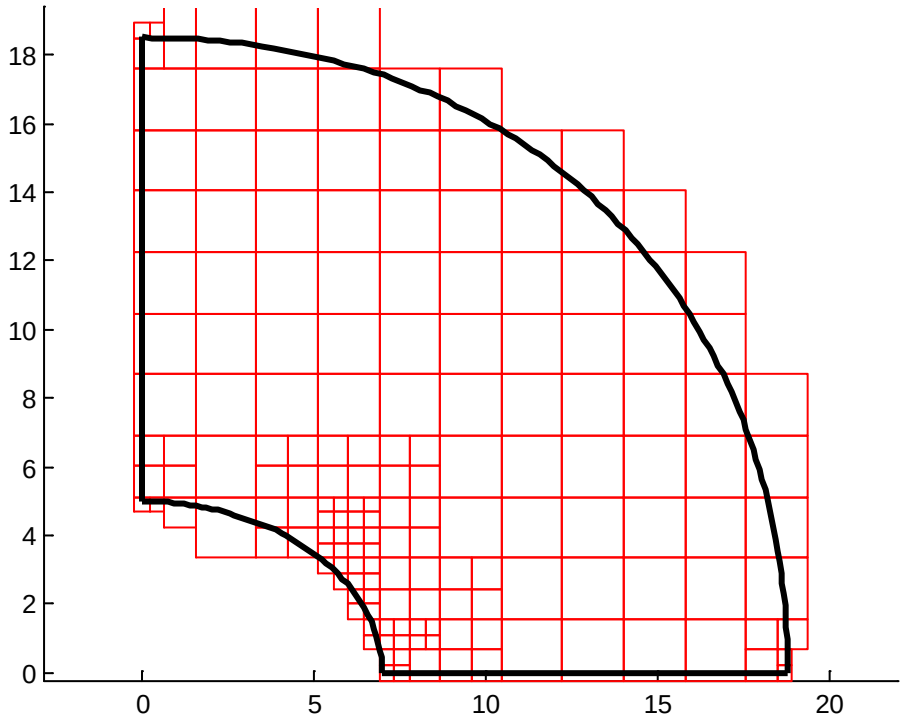

Figura 19. Geometría solución del problema Cilindro 3.

Segunda ejecución del problema.

Se propone una segunda ejecución del problema con los mismos parámetros que la primera, pero con la disposición de los datos almacenados en ésta.

Se llega a la siguiente solución:

- Valor de las variables de diseño: [18’8048, 13’4874, 13’4874, 18’5099] m.
- Volumen: 252’301 m^3.

Con:

- 301 geometrías consideradas en el proceso.
- 16 geometrías analizadas con MEF.
- 285 geometrías reutilizadas.
- Tiempo promedio de análisis MEF: 113’6698 seg.

El tiempo de análisis del problema es: 9 h 30 min 14 seg.

El tiempo de iteración del problema es: 48 min 56 seg.

La geometría es idéntica a la de la primera ejecución.

Tercera ejecución del problema.

La tercera ejecución del problema se realiza con los mismos parámetros que la primera salvo el número de puntos de inicio, que es el doble (40), y con la disposición de los datos almacenados en ésta.

Se llega a la siguiente solución:

- Valor de las variables de diseño: [18’8048, 13’4874, 13’4874, 18’5099] m.
- Volumen: 252’301 m^3.

Con:

- 381 geometrías consideradas en el proceso.
- 36 geometrías analizadas con MEF.
- 345 geometrías reutilizadas.
- Tiempo promedio de análisis MEF: 131’3751 seg.

El tiempo de análisis del problema es: 13 h 54 min 13 seg.

El tiempo de iteración del problema es: 1 h 55 min 24 seg.

La geometría es idéntica a la de la primera ejecución.

Recopilando los tiempos obtenidos:

Tabla 6. Comparación de tiempos del problema Cilindro 3.

Cilindro 3	**1ª ejecución 20 puntos**	**2ª ejecución 20 puntos**	**3ª ejecución 40 puntos**
Tiempo de análisis	9 h 45 min 42 seg	9 h 30 min 14 seg	13 h 54 min 13 seg
Tiempo de iteración	2 h 20 min 54 seg	48 min 56 seg	1 h 55 min 24 seg
Ahorro de tiempo	7 h 24 min 48 seg	8 h 41 min 18 seg	11 h 58 min 49 seg
Ahorro de tiempo (porcentaje)	75’94%	91’42%	86’17%

Las conclusiones que se sacan de los resultados son:

- El uso de datos almacenados supone un ahorro significativo de tiempo desde la primera ejecución del problema.
- Los tiempos de iteración de los dos últimos análisis evidencian un número no muy elevado de usos de datos almacenados; hay mucha variedad en los valores sugeridos de las variables de diseño por el proceso de optimización.
- La gran diferencia entre los tiempos de análisis de las ejecuciones de 20 puntos de inicio y la de 40 demuestra que el problema tiene muchos mínimos locales entre los que buscar el global.

4.3.2. Uso de resultados almacenados y redes neuronales

En este subapartado se estudia el efecto de utilizar ambas mejoras propuestas al mismo tiempo.

Primera ejecución del problema.

La primera ejecución del problema (esto es, sin disponibilidad alguna de datos almacenados de ejecuciones anteriores) se realiza con los mismos parámetros de la primera ejecución del subapartado anterior, y con los siguientes parámetros relacionados con las redes neuronales:

- 30 datos MEF como mínimo para la construcción de las redes neuronales.
- 4 redes neuronales para cada conjunto de datos (volumen-gradiente del volumen, restricción-gradiente de la restricción) se construyen en un bucle para la elección de las que tengan el menor error de validación.
- 5 usos consecutivos de las redes neuronales para hacer cálculos.
- 5 análisis MEF se realizan entre la utilización de un conjunto de redes neuronales y el siguiente.

Se llega a la siguiente solución:

- Valor de las variables de diseño: [18’8048, 13’4874, 13’4874, 18’5099] m.
- Volumen: 252’301 m^3.

Con:

- 196 geometrías consideradas en el proceso.
- 40 geometrías analizadas con MEF.
- 11 geometrías analizadas con redes neuronales.
- 145 geometrías reutilizadas.

El tiempo de análisis del problema es: 5 h 18 min 38 seg.

El tiempo de iteración del problema es: 1 h 13 min 9 seg.

La geometría es idéntica a la del subapartado anterior.

Segunda ejecución del problema.

La segunda ejecución del problema se realiza con los mismos parámetros que la primera, pero con la disposición de los datos almacenados en ésta.

Se llega a la siguiente solución:

- Valor de las variables de diseño: [18’8048, 13’4874, 13’4874, 18’5099] m.
- Volumen: 252’301 m^3.

Con:

- 182 geometrías consideradas en el proceso.
- 10 geometrías analizadas con MEF.
- 11 geometrías analizadas con redes neuronales.
- 161 geometrías reutilizadas.

El tiempo de análisis del problema es: 4 h 8 seg.

El tiempo de iteración del problema es: 27 min 10 seg.

La geometría es idéntica a la de la primera ejecución.

Tercera ejecución del problema.

La tercera ejecución del problema se realiza con los mismos parámetros que la primera salvo el número de puntos de inicio, que es el doble (40), y con la disposición de los datos almacenados en ésta.

Se llega a la siguiente solución:

- Valor de las variables de diseño: [18’8048, 13’4874, 13’4874, 18’5099] m.
- Volumen: 252’301 m^3.

Con:

- 260 geometrías consideradas en el proceso.

- 20 geometrías analizadas con MEF.
- 21 geometrías analizadas con redes neuronales.
- 219 geometrías reutilizadas.

El tiempo de análisis del problema es: 7 h 3 min 29 seg.

El tiempo de iteración del problema es: 1 h 10 min 38 seg.

La geometría es idéntica a la de la primera ejecución.

Recopilando los tiempos obtenidos:

Tabla 7. Comparación de tiempos del problema Cilindro 3 con estrategia NNet.

Cilindro 3	**1ª ejecución 20 puntos**	**2ª ejecución 20 puntos**	**3ª ejecución 40 puntos**
Tiempo de análisis	5 h 18 min 38 seg	4 h 8 seg	7 h 3 min 29 seg
Tiempo de iteración	1 h 13 min 9 seg	27 min 10 seg	1 h 10 min 38 seg
Ahorro de tiempo	4 h 5 min 29 seg	3 h 32 min 58 seg	5 h 52 min 51 seg
Ahorro de tiempo (porcentaje)	77’04%	88’69%	83’32%

La utilización de redes neuronales hace más difícil sacar conclusiones precisas de los tiempos de análisis, ya que el azar influye considerablemente (sobre todo en el tiempo de entrenamiento de las redes neuronales, que depende de valores iniciales aleatorios, y en los valores de las variables de diseño sugeridos por el proceso de optimización). Con todo, se puede concluir que:

- El uso de datos almacenados y redes neuronales supone un ahorro significativo de tiempo desde la primera ejecución del problema.
- Los tiempos de iteración de los dos últimos análisis muestran un número no muy elevado de usos de datos almacenados; hay mucha variedad en los valores sugeridos de las variables de diseño por el proceso de optimización.

- La diferencia entre los tiempos de análisis de la 1ª y la 2ª ejecución con 20 puntos de inicio evidencia un uso mayor de cálculos con redes neuronales en la última, ya que ésta dispone desde el principio de los datos de análisis MEF de la ejecución anterior.
- La gran diferencia entre los tiempos de análisis de las ejecuciones de 20 puntos de inicio y la de 40 demuestra que el problema tiene muchos mínimos locales entre los que buscar el global.

A continuación se presenta la comparación de tiempos definitiva del problema: los tiempos de análisis cuando sólo se utiliza la repetición de datos, y los tiempos de iteración cuando se utilizan análisis MEF y redes neuronales. De esta forma se comprueba el ahorro de tiempo total entre el tiempo de ejecución del anterior programa y el del nuevo cuando éste utiliza todo su potencial:

Tabla 8. Comparación de tiempos definitiva del problema Cilindro 3.

Cilindro 3	**1ª ejecución 20 puntos**	**2ª ejecución 20 puntos**	**3ª ejecución 40 puntos**
Tiempo de análisis (sólo MEF)	9 h 45 min 42 seg	9 h 30 min 14 seg	13 h 54 min 13 seg
Tiempo de iteración MEF + NNet	1 h 13 min 9 seg	27 min 10 seg	1 h 10 min 38 seg
Ahorro de tiempo total	8 h 32 min 33 seg	9 h 3 min 4 seg	12 h 44 min 25 seg
Ahorro de tiempo total (porcentaje)	87’51%	95’24%	91’63%

El ahorro de tiempo podría haber sido aún mayor si se hubiera empleado una estrategia de redes neuronales más agresiva (por ejemplo, mediante el aumento del número de usos consecutivos de las redes neuronales para hacer cálculos).

4.4. PROBLEMA PRESA

Este problema consiste en una presa de gravedad sometida a la presión hidrostática del agua con hueco interno (galería) a optimizar manteniendo el contorno externo.

Se consideran diez variables de diseño.

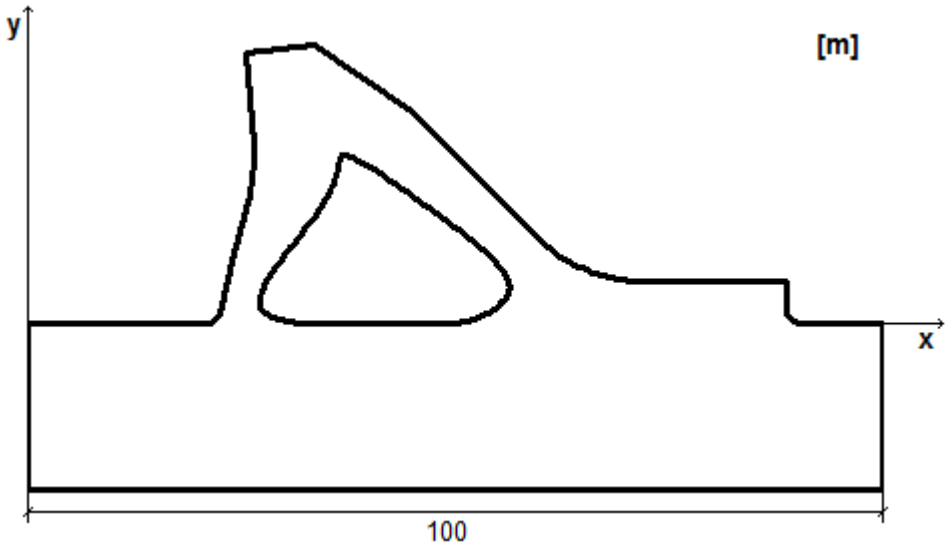

Figura 20. Geometría inicial del problema Presa.

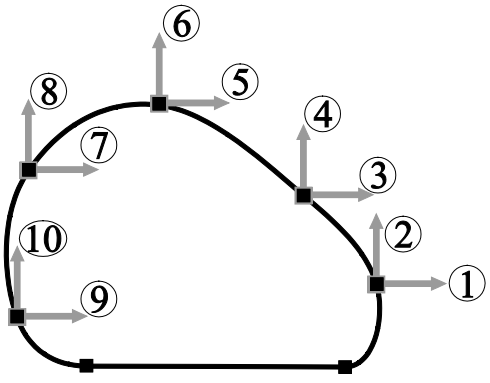

Figura 21. Variables de diseño del problema Presa

4.4.1. Uso de resultados almacenados

En este subapartado se estudia el efecto de la mejora que permite la utilización de resultados almacenados en la ejecución en curso y en anteriores ejecuciones del problema.

Primera ejecución del problema.

La primera ejecución de este problema (esto es, sin disponibilidad alguna de datos almacenados de ejecuciones anteriores) se realiza con:

- 20 puntos de inicio potenciales.
- 1 restricción (tensión máxima admisible de Von Mises 27'5*10^6 Pa).
- Elementos cuadráticos.
- Error objetivo en norma energética: 1%.
- Número máximo de iteraciones del proceso de mallado h-adaptativo: 3.
- Tolerancias del criterio de parada (tanto $TolX$ como $TolFun$): 10^{-3}. Son tolerancias absolutas de las que se valdrá el programa para saber que ha llegado a la solución. Si la norma de $(X_i - X_{i+1})$ es menor que $TolX$ y la norma de $(f(X_i) - f(X_{i+1}))$ es menor que $TolFun$, el programa considera que se ha llegado a la solución. $TolX$ es 10^{-3} m y $TolFun$ es 10^{-3} m^3.

Se llega a la siguiente solución:

- Valor de las variables de diseño: [60, 3, 50, 17, 37'9475, 28, 27, 26, 24'336, 3] m.
- Volumen: 2374'93 m^3.

Con:

- 173 geometrías consideradas en el proceso.
- 28 geometrías analizadas con MEF.
- 145 geometrías reutilizadas.
- Tiempo promedio de análisis MEF: 1829'4284 seg.

El tiempo de análisis del problema es: 3 d 15 h 54 min 51 seg.

El tiempo de iteración del problema es: 14 h 22 seg.

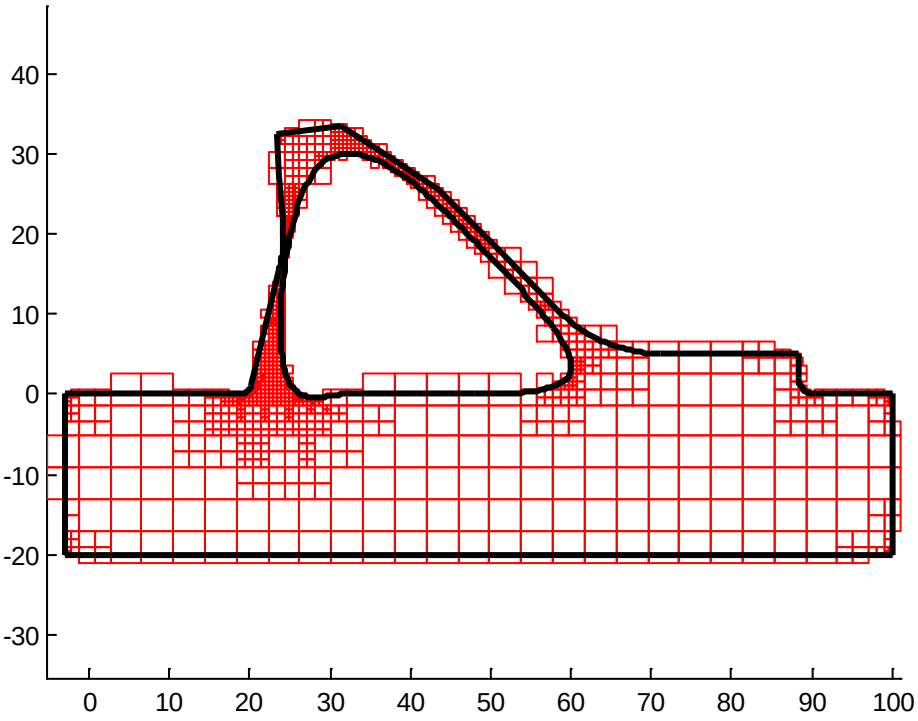

Figura 22. Geometría solución del problema Presa.

Segunda ejecución del problema.

Se propone una segunda ejecución del problema con los mismos parámetros que la primera, pero con la disposición de los datos almacenados en ésta.

Se llega a la siguiente solución:

- Valor de las variables de diseño: [60, 3, 50, 17, 37’9475, 28, 27, 26, 24’336, 3] m.
- Volumen: 2374’93 m^3.

Con:

- 163 geometrías consideradas en el proceso.
- 19 geometrías analizadas con MEF.
- 144 geometrías reutilizadas.
- Tiempo promedio de análisis MEF: 1943’6424 seg.

El tiempo de análisis del problema es: 3 d 16 h 13 seg.

El tiempo de iteración del problema es: 10 h 15 min 20 seg.

La geometría es idéntica a la de la primera ejecución.

Tercera ejecución del problema.

La tercera ejecución del problema se realiza con los mismos parámetros que la primera salvo el número de puntos de inicio, que es el doble (40), y con la disposición de los datos almacenados en ésta.

Se llega a la siguiente solución:

- Valor de las variables de diseño: [60, 3, 50, 17, 37’9475, 28, 27, 26, 24’336, 3] m.
- Volumen: 2374’93 m^3.

Con:

- 243 geometrías consideradas en el proceso.
- 35 geometrías analizadas con MEF.
- 208 geometrías reutilizadas.
- Tiempo promedio de análisis MEF: 1430’3374 seg (el tiempo es tan inferior a las anteriores ejecuciones porque 12 de las geometrías analizadas en esta ejecución han dado lugar a errores, terminando abruptamente y antes de tiempo los análisis MEF correspondientes).

El tiempo de análisis del problema es: 4 d 32 min 51 seg.

El tiempo de iteración del problema es: 12 h 12 min 37 seg.

La geometría es idéntica a la de la primera ejecución.

Recopilando los tiempos obtenidos:

Tabla 9. Comparación de tiempos del problema Presa.

Presa	**1ª ejecución 20 puntos**	**2ª ejecución 20 puntos**	**3ª ejecución 40 puntos**
Tiempo de análisis	3 d 15 h 54 min 51 seg	3 d 16 h 13 seg	4d 32 min 51 seg
Tiempo de iteración	14 h 22 seg	10 h 15 min 20 seg	12 h 12 min 37 seg
Ahorro de tiempo	3 d 1 h 54 min 29 seg	3 d 5 h 44 min 53 seg	3 d 12 h 20 min 14 seg
Ahorro de tiempo (porcentaje)	84’07%	88’35%	87’35%

Las conclusiones que se sacan de los resultados son:

- El uso de datos almacenados supone un ahorro significativo de tiempo desde la primera ejecución del problema.
- Los tiempos de iteración de los dos últimos análisis evidencian un número no muy elevado de usos de datos almacenados; hay mucha variedad en los valores sugeridos de las variables de diseño por el proceso de optimización.
- La gran diferencia entre los tiempos de análisis de las ejecuciones de 20 puntos de inicio y la de 40 demuestra que el problema tiene muchos mínimos locales entre los que buscar el global.

4.4.2. Uso de resultados almacenados y redes neuronales

En este subapartado se estudia el efecto de utilizar ambas mejoras propuestas al mismo tiempo.

Primera ejecución del problema.

La primera ejecución del problema (esto es, sin disponibilidad alguna de datos almacenados de ejecuciones anteriores) se realiza con los mismos parámetros de la primera ejecución del subapartado anterior, y con los siguientes parámetros relacionados con las redes neuronales:

- 20 datos MEF como mínimo para la construcción de las redes neuronales.
- 5 redes neuronales para cada conjunto de datos (volumen-gradiente del volumen, restricción-gradiente de la restricción) se construyen en un bucle para la elección de las que tengan el menor error de validación.
- 5 usos consecutivos de las redes neuronales para hacer cálculos.
- 5 análisis MEF se realizan entre la utilización de un conjunto de redes neuronales y el siguiente.

Se llega a la siguiente solución:

- Valor de las variables de diseño: [60, 3, 50, 17, 37’9475, 28, 27, 26, 24’336, 3] m.
- Volumen: 2374’93 m^3.

Con:

- 150 geometrías consideradas en el proceso.
- 24 geometrías analizadas con MEF.
- 6 geometrías analizadas con redes neuronales.
- 120 geometrías reutilizadas.

El tiempo de análisis del problema es: 2 d 7 h 38 min 38 seg.

El tiempo de iteración del problema es: 11 h 23 min 14 seg.

La geometría es idéntica a la del subapartado anterior.

Segunda ejecución del problema.

La segunda ejecución del problema se realiza con los mismos parámetros que la primera, pero con la disposición de los datos almacenados en ésta.

Se llega a la siguiente solución:

- Valor de las variables de diseño: [60, 3, 50, 17, 37’9475, 28, 27, 26, 24’336, 3] m.
- Volumen: 2374’93 m^3.

Con:

- 152 geometrías consideradas en el proceso.
- 10 geometrías analizadas con MEF.
- 11 geometrías analizadas con redes neuronales.
- 131 geometrías reutilizadas.

El tiempo de análisis del problema es: 1 d 20 h 58 min 46 seg.

El tiempo de iteración del problema es: 4 h 56 min 13 seg.

La geometría es idéntica a la de la primera ejecución.

Tercera ejecución del problema.

La tercera ejecución del problema se realiza con los mismos parámetros que la primera salvo el número de puntos de inicio, que es el doble (40), y con la disposición de los datos almacenados en ésta.

Se llega a la siguiente solución:

- Valor de las variables de diseño: [60, 3, 50, 17, 37’9475, 28, 27, 26, 24’336, 3] m.
- Volumen: 2374’93 m^3.

Con:

- 238 geometrías consideradas en el proceso.
- 20 geometrías analizadas con MEF.
- 21 geometrías analizadas con redes neuronales.
- 197 geometrías reutilizadas.

El tiempo de análisis del problema es: 3 d 7 h 41 min 31 seg.

El tiempo de iteración del problema es: 10 h 36 min 26 seg.

La geometría es idéntica a la de la primera ejecución.

Recopilando los tiempos obtenidos:

Tabla 10. Comparación de tiempos del problema Presa con estrategia NNet.

Presa	**1ª ejecución 20 puntos**	**2ª ejecución 20 puntos**	**3ª ejecución 40 puntos**
Tiempo de análisis	2 d 7 h 38 min 38 seg	1 d 20 h 58 min 46 seg	3 d 7 h 41 min 31 seg
Tiempo de iteración	11 h 23 min 14 seg	4 h 56 min 13 seg	10 h 36 min 26 seg
Ahorro de tiempo	1 d 20 h 15 min 24 seg	1 d 16 h 2 min 33 seg	2 d 21 h 5 min 5 seg
Ahorro de tiempo (porcentaje)	79’54%	89’02%	86’69%

La utilización de redes neuronales hace más difícil sacar conclusiones precisas de los tiempos de análisis, ya que el azar influye considerablemente (sobre todo en el tiempo de entrenamiento de las redes neuronales, que depende de valores iniciales aleatorios, y en los valores de las variables de diseño sugeridos por el proceso de optimización). Con todo, se puede concluir que:

- El uso de datos almacenados y redes neuronales supone un ahorro significativo de tiempo desde la primera ejecución del problema.
- Los tiempos de iteración de los dos últimos análisis muestran un número no muy elevado de usos de datos almacenados; hay mucha variedad en los valores sugeridos de las variables de diseño por el proceso de optimización.
- La diferencia entre los tiempos de análisis de la 1ª y la 2ª ejecución con 20 puntos de inicio evidencia un uso mayor de cálculos con redes neuronales en la última, ya que ésta dispone desde el principio de los datos de análisis MEF de la ejecución anterior.
- La gran diferencia entre los tiempos de análisis de las ejecuciones de 20 puntos de inicio y la de 40 demuestra que el problema tiene muchos mínimos locales entre los que buscar el global.

A continuación se presenta la comparación de tiempos definitiva del problema: los tiempos de análisis cuando sólo se utiliza la repetición de datos, y los tiempos de iteración cuando se utilizan análisis MEF y redes neuronales. De esta forma se comprueba el ahorro de tiempo total entre el tiempo de ejecución del anterior programa y el del nuevo cuando éste utiliza todo su potencial:

Tabla 11. Comparación de tiempos definitiva del problema Presa.

Presa	**1ª ejecución 20 puntos**	**2ª ejecución 20 puntos**	**3ª ejecución 40 puntos**
Tiempo de análisis (sólo MEF)	3 d 15 h 54 min 51 seg	3 d 16 h 13 seg	4 d 32 min 51 seg
Tiempo de iteración MEF + NNet	11 h 23 min 14 seg	4 h 56 min 13 seg	10 h 36 min 26 seg
Ahorro de tiempo total	3 d 4 h 31 min 37 seg	3 d 11 h 4 min	3 d 13 h 56 min 25 seg
Ahorro de tiempo total (porcentaje)	87’05%	94’39%	89’01%

El ahorro de tiempo podría haber sido aún mayor si se hubiera empleado una estrategia de redes neuronales más agresiva (por ejemplo, mediante el aumento del número de usos consecutivos de las redes neuronales para hacer cálculos).

5. CONCLUSIONES Y TRABAJOS FUTUROS

Se han cumplido todos los objetivos propuestos:

- Facilitar la utilización del programa al usuario:
  - Los parámetros de un problema se definen en un módulo separado del resto de módulos del programa.
  - Se ha implementado un proceso de tratamiento de posibles errores para evitar la interrupción del programa y presentar la información de las causas de dichos errores al usuario.
  - El programa proporciona al usuario una gran cantidad de información útil, y almacena los datos calculados de un determinado problema para su uso en futuras ejecuciones del mismo.
  - Se ha desarrollado un manual del usuario claro y conciso.

- Facilitar la edición del programa al programador:
  - En el nuevo programa hay un único conjunto de tres módulos principales donde se encuentra el código de éste.
  - El código de cada módulo está organizado en “bloques” con propósitos concretos y se han eliminado líneas de código innecesarias.
  - Se ha desarrollado un manual del programador muy detallado.

- Reducir el tiempo de ejecución del programa: el uso de datos almacenados y redes neuronales consigue un ahorro de tiempo muy significativo; dependiendo del problema, pueden ser horas o incluso días.

Se sugieren las siguientes mejoras que se podrían implementar:

- Realización de una interfaz de usuario (GUI) que permita la definición de los parámetros de un problema sin necesidad de abrir el módulo correspondiente en el editor de Matlab.
- Elaboración de nuevas estrategias para definir el conjunto de datos que se usarán en la construcción de las redes neuronales.

BIBLIOGRAFÍA

[1] Gabriel Bugeda, Juan José Ródenas García, Eugenio Oñate, “An integration of a low cost adaptative remeshing strategy in the solution of structural shape optimization problems using evolutionary methods”, CAS 4108, Computers and Structures 2007.

[2] Luis Esparcia García, “Desarrollo de una metodología eficiente de optimización de forma basado en algoritmos de tipo gradiente y un código MEF de mallados cartesianos independientes de la geometría”, Trabajo fin de Máster, Universidad Politécnica de Valencia, Julio de 2014.

[3] Javier Fuenmayor Fernández, Juan José Ródenas García, José Enrique Tarancón Caro, Manuel Tur Valiente, Ana Vercher martínez, “Cálculo Estructural, Método de los Elementos Finitos”, Apuntes de la asignatura, Universidad Politécnica de Valencia, 2013.

[4] Mark Hudson Beale, Martin T. Hagan, Howard B. Demuth, “Neural Network Toolbox™”, Matlab User’s Guide, 2015.

[5] Onofre Marco, “A shape sensitivity analysis module with geometric representation by nurbs for a 2D finite element program based on cartesian meshes independent of the geometry”, Tesis de Máster, Universidad Politécnica de Valencia, Julio 2012.

[6] Elke Pahl, “Optimization. An attempt at describing the State of the Art”, Informe del Centro Internacional de Métodos Numéricos en Ingeniería (CIMNE), Barcelona, Marzo 2004.

[7] Juan José Ródenas García, “Error de discretización en el cálculo de sensibilidades mediante el método de los elementos finitos”, Tesis Doctoral, Universidad Politécnica de Valencia, 2001.

[8] http://es.mathworks.com/help/optim/ug/fgoalattain.html

[9] http://es.mathworks.com/help/optim/ug/fmincon.html

[10] http://es.mathworks.com/help/optim/ug/fminimax.html

[11] http://es.wikipedia.org/wiki/MATLAB

APÉNDICES

APÉNDICE A. MANUAL DEL USUARIO

En este apéndice se explican, con todo detalle, los pasos que ha de seguir un usuario a la hora de usar el programa para resolver un problema determinado.

En el momento de la redacción de este manual no hay una interfaz gráfica (GUI) implementada, por lo que esta guía describe la estructura de directorios del programa que todo usuario debe conocer a la hora de utilizarlo y el archivo en el que se definen los parámetros del problema.

Por último, se desarrolla un ejemplo sencillo guiado, donde se muestra la forma de acceder a toda la información que el programa proporciona al usuario.

A.1. DIRECTORIOS

La carpeta maestra, que contiene todos los archivos y subcarpetas del programa (llamada “Programa optimizador de forma 2D” en el momento de la redacción), debe ser el directorio establecido en matlab a la hora de ejecutar el programa.

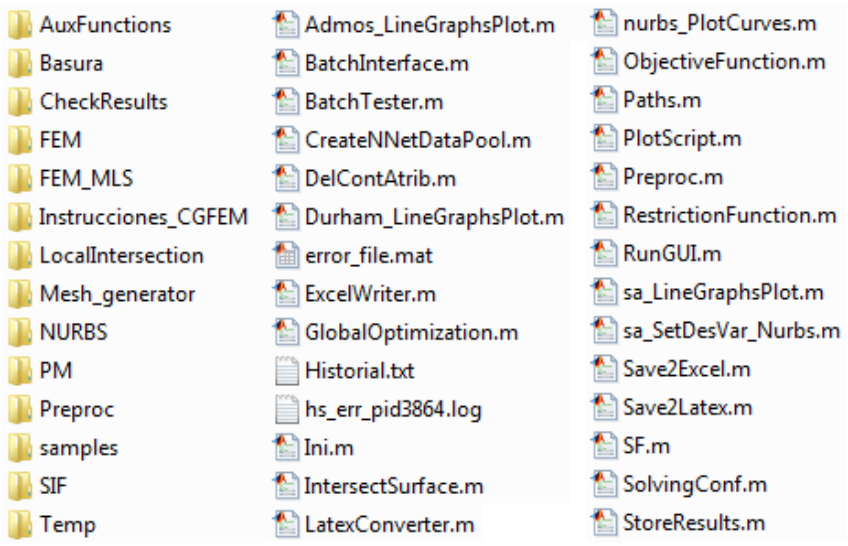

Figura 23. Contenido de la carpeta maestra.

En la carpeta *samples* se almacena la lista de problemas que se pueden resolver.

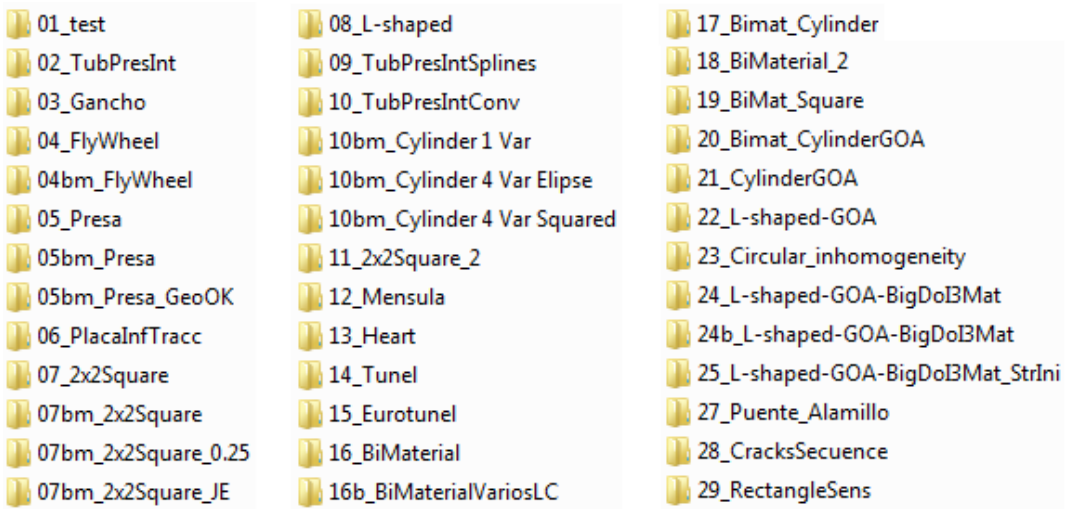

Figura 24. Contenido de la carpeta *samples*.

Dentro de la carpeta de cada problema hay archivos necesarios para el programa. También, si el programa las ha creado, están presentes las carpetas donde se almacenarán los datos provenientes de las ejecuciones del problema: *Results* y *Stored ResultData*.

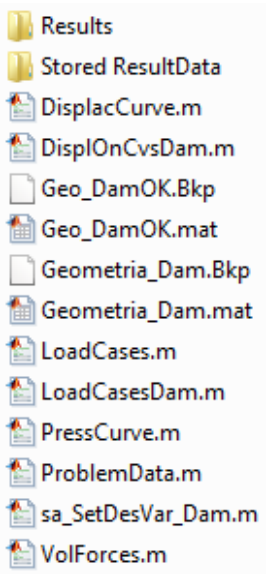

Figura 25. Contenido de la carpeta *05bm_Presa_GeoOK*.

A.2. ARCHIVO *PROBLEMDATA.M*

Es aquí donde el usuario define los parámetros del problema que va a resolver. Al no existir una interfaz gráfica, deberá escribir los datos directamente en las líneas de código:

```
'IniValRes'

    IniValRes.x0= #Vector de reales;
    IniValRes.Aineq= #Matriz de reales;
    IniValRes.bineq= #Vector de reales;
    IniValRes.Aeq= #Matriz de reales;
    IniValRes.beq= #Vector de reales;
    IniValRes.lb= #Vector de reales;
    IniValRes.ub= #Vector de reales;
```

En la variable global *IniValRes* se definen:

- Los valores iniciales de las variables de diseño (en el campo *x0*).
- Las restricciones de desigualdad entre ellas (en los campos *Aineq*, *bineq*).
- Las restricciones de igualdad entre ellas (si las hubiera, en los campos *Aeq*, *beq*).
- Los límites inferiores de las variables de diseño (en el campo *lb*).
- Los límites superiores de las variables de diseño (en el campo *ub*).

```
'RunOptions'

    RunOptions.NumTrialPoints= #Entero;
    RunOptions.NumStageOnePoints= #Entero;

    RunOptions.Algorithm= 'sqp'/'interior-point';
    RunOptions.Display= 'off'/'iter'/'final'/'notify';
    RunOptions.TolFun= #Real;
    RunOptions.TolX= #Real;
    RunOptions.UseParallel= false/true;
```

En la variable global *RunOptions* se definen varias opciones del proceso de optimización:

- El número de puntos de prueba a partir de los cuales el programa intentará encontrar mínimos locales (en los campos *NumTrialPoints*, *NumStageOnePoints*).
- El algoritmo utilizado en esa tarea (en el campo *Algorithm*).
- El modo de presentación de resultados (en el campo *Display*).
- Las tolerancias (absolutas) de las que se valdrá el programa para saber que ha llegado a la solución (en los campos *TolFun*, *TolX*). Si la norma de (X_i-X_{i+1}) es menor que *TolX* y la norma de ($f(X_i)$-$f(X_{i+1})$) es menor que *TolFun*, el programa considera que se ha llegado a la solución.
- La utilización de la computación en paralelo (en el campo *UseParallel*).

```
RunOptions.UsePrevData= 0/1;

RunOptions.UseNNet= 0/1;
```

El usuario decide si desea o no utilizar datos almacenados de análisis anteriores del mismo problema (en el campo *UsePrevData*), así como el uso de las redes neuronales en la realización de cálculos (en el campo *UseNNet*).

```
RunOptions.DataPoolOrigin= 1;

RunOptions.VolNNetnumNeurons= #Entero;
RunOptions.ConstrNNetnumNeurons= #Entero;

RunOptions.NNetTrainFcn= 'trainlm'/'trainscg';
RunOptions.NNetminInputs= #Entero;
RunOptions.NNetnumLoop= #Entero;
RunOptions.maxNNetUses= #Entero;
RunOptions.maxFEMUses= #Entero;
```

Se definen parámetros relacionados con las redes neuronales:

- La estrategia a la hora de construir el conjunto de datos que se usarán para crear las redes neuronales (en el momento de la redacción de este manual sólo hay una implementada, luego el valor del campo *DataPoolOrigin* sólo puede ser 1).

- El número de neuronas con las que se construirán las redes neuronales del volumen y las restricciones (en los campos *VolNNetnumNeurons* y *ConstrNNetnumNeurons* respectivamente).
- La función que se usará para el entrenamiento de las redes neuronales (en el campo *NNetTrainFcn*).
- El número mínimo de datos provenientes de análisis MEF diferentes que se requiere para la creación de las redes (en el campo *NNetminInputs*).
- Cuando una red neuronal se crea, se le asignan valores aleatorios de pesos y tendencias, luego es conveniente crear varias y quedarse finalmente con la que tenga menor error de validación. En el campo *NnetnumLoop* se define el número de redes que se crearán en un bucle con esa finalidad.
- El número máximo de usos consecutivos de unas redes determinadas (en el campo *maxNNetUses*).
- El número máximo de análisis MEF entre el fin de la utilización de un conjunto de redes y la construcción del siguiente (en el campo *maxFEMUses*).

```
RunOptions.NumConstr= #Entero;
```

Se define el número de restricciones que tiene en cuenta el proceso de optimización para alcanzar la solución. Debe coindicir con las dimensiones de los vectores correspondientes en los otros módulos. Si el usuario quiere considerar más de una restricción, debe editar los módulos del programa, por lo que la consulta al manual del programador es obligatoria.

```
RunOptions.ScaledAxisXmin= 0;
RunOptions.ScaledAxisXmax= max(IniValRes.ub)+…
    min(IniValRes.lb);
RunOptions.ScaledAxisYmin= 0;
RunOptions.ScaledAxisYmax= max(IniValRes.ub)+…
    min(IniValRes.lb);
```

Se definen los ejes fijos que se utilizarán para representar las geometrías correspondientes. El usuario puede darles el valor deseado; el existente es sólo una

sugerencia (desde el origen de los ejes hasta la suma del valor máximo de los límites superiores de las variables y el valor mínimo de los límites inferiores de las variables).

```
'StartGeo'

    load('Geometria_NurbsOpt_4var_squared.mat');

    Geo.Cv(1).FixedCv=0;
    Geo.Cv(2).FixedCv=0;
    Geo.Cv(3).FixedCv=0;

    Geo.Material.MaxStress= #Real;
    Geo.Material.ElastModul= #Real;
    Geo.Material.StressFOS= #Real;
```

Se carga el archivo (creado previamente en cgFEM) que contiene la geometría base del problema y los datos de las propiedades del material (en este caso es *Geometria_NurbsOpt_4var_squared.mat*). Se indican las curvas de la geometría que no son fijas (*FixedCv=0*) y se modifican a voluntad algunos parámetros del material:

- Tensión máxima admisible (en el campo *MaxStress*).
- Módulo de elasticidad (en el campo *ElastModul*).
- Factor de seguridad de las tensiones (en el campo *StressFOS*).

```
'ProcessGeo'

    Geo.Pt.XYZ(2,4)=Reference.Parameters(4);
    Geo.Pt.XYZ(1,6)=Reference.Parameters(2);
    Geo.Pt.XYZ(2,6)=Reference.Parameters(3);
    Geo.Pt.XYZ(1,3)=Reference.Parameters(1);

    SensAnaGlob.Param.DesVarFile=…
        'sa_SetDesVar_NurbsOpt_4var.m';
```

En el proceso de análisis MEF, se asignan las coordenadas de determinados puntos de la geometría a las variables de diseño correspondientes según lo definido aquí por el usuario.

Se establece la función encargada de modificar la geometría con las variables de diseño (en el campo *DesVarFile*).

```
'MeshParam'

    BatchInfo.HadaptiveMethod=1;
    BatchInfo.AnalysisType='h_adaptive_mesh';
    BatchInfo.EType=1/2;
    BatchInfo.IniMesh=5;
    BatchInfo.IntsMesh=5;
    BatchInfo.MaxLevelMesh=11;
    BatchInfo.SquareMesh=1;
    BatchInfo.UniformRefinement=0;
    BatchInfo.RecovType=300;
    BatchInfo.TargErr=[-1 -1 -1 -1 #Real -1];
    BatchInfo.ErrReduct=50;
    BatchInfo.MxRefInc=3;
    BatchInfo.MxRefLevel=11;
    BatchInfo.MxNIter=#Entero;
    BatchInfo.GeometricalRefinamentArea=0;
    BatchInfo.GeometricalRefinamentCurvature=1;
    BatchInfo.ExactPressSolution=0;
    BatchInfo.AnalysisMode2D=PlaneStrain;
    BatchInfo.LoadCase2Solve=[1 2 3 4 5 6];
    BatchInfo.Error_Control=0;
    BatchInfo.ParTrans=Quadratic;
    BatchInfo.TerrorTerm=0;
    BatchInfo.MeshCoords=#Matriz de reales;
```

Se definen todos los parámetros necesarios para el proceso de mallado del bloque de análisis MEF. Los más significativos son:

- El tipo de elemento (en el campo *EType*). Puede ser lineal (1) o cuadrático (2).
- El máximo error en norma energética deseado en la malla (en el campo *TargErr*). Los valores -1 indican que esos casos de carga no sirven para refinar la malla; los primeros están relacionados con los campos de velocidades de las, en este caso, cuatro variables de diseño, y el último con las sensibilidades.
- El máximo número de iteraciones de la malla en el proceso h-adaptativo (en el campo *MxNIter*).
- Las coordenadas de la malla (en el campo *MeshCoords*). La malla debe cubrir toda la geometría.

A.3. DESARROLLO DE EJEMPLO GUIADO

A continuación se expone el proceso de resolución del problema *10bm_Cylinder 4 Var Squared* de la carpeta *samples*.

La geometría inicial y las variables de diseño a tratar son las siguientes:

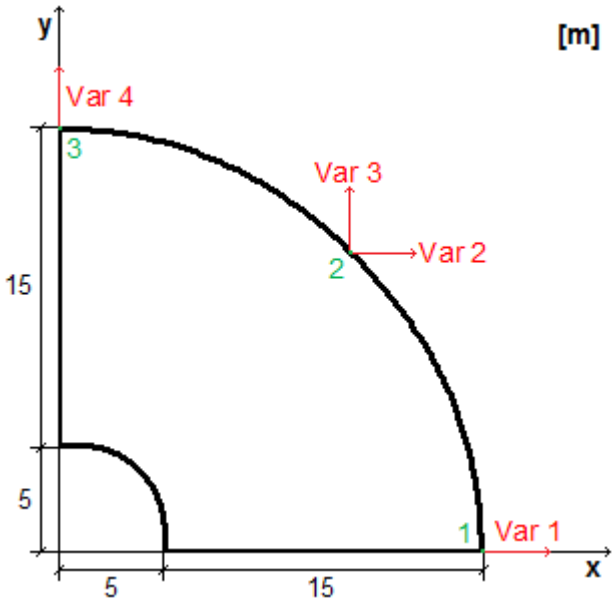

Figura 26. Geometría inicial y variables de diseño del problema *10bm_Cylinder 4 Var Squared*

Se tiene que:

- La variable 1 es la coordenada “x” del punto 1.
- La variable 2 es la coordenada “x” del punto 2.
- La variable 3 es la coordenada “y” del punto 2.
- La variable 4 es la coordenada “y” del punto 3.

El primer paso siempre es definir los parámetros del problema. Se inicia Matlab, y se abre el archivo `ProblemData.m` de la carpeta del problema deseado:

```
'IniValRes'

    IniValRes.x0=[20 14 14 20];
    IniValRes.Aineq=[-1 0 1 0; 0 1 0 -1];
    IniValRes.bineq=[-0.5; -0.5];
    IniValRes.Aeq=[];
    IniValRes.beq=[];
    IniValRes.lb=[10 7 7 10];
    IniValRes.ub=[25 25 25 25];
```

Se establecen:

- Los valores iniciales de las variables de diseño (en el campo *x0*): 20 m para las variables 1 y 4, 14 m para las variables 2 y 3.
- Las restricciones de desigualdad entre ellas (en los campos *Aineq*, *bineq*), de la siguiente forma:

$$Var2 + 0'5 \leq Var1 \;\rightarrow\; -Var1 + Var2 \leq -0'5$$

La coordenada "x" del punto 2 se mantendrá siempre a una distancia de 0'5 m como mínimo de la coordenada "x" del punto 1.

$$Var3 + 0'5 \leq Var4 \;\rightarrow\; Var3 - Var4 \leq -0'5$$

La coordenada "y" del punto 2 se mantendrá siempre a una distancia de 0'5 m como mínimo de la coordenada "y" del punto 3.

Se tiene que:

$$\begin{pmatrix} -1 & 0 & 1 & 0 \\ 0 & 1 & 0 & -1 \end{pmatrix} * \begin{pmatrix} Var1 \\ Var3 \\ Var2 \\ Var4 \end{pmatrix} \leq \begin{pmatrix} -0'5 \\ -0'5 \end{pmatrix} \rightarrow Aineq * \begin{pmatrix} Var1 \\ Var3 \\ Var2 \\ Var4 \end{pmatrix} \leq bineq$$

- Las restricciones de igualdad entre ellas (en los campos *Aeq*, *beq*). En este problema no las hay, si bien se definirían análogamente a las de desigualdad.
- Los límites inferiores de las variables de diseño (en el campo *lb*): 10 m para las variables 1 y 4, 7 m para las variables 2 y 3.
- Los límites superiores de las variables de diseño (en el campo *ub*): 25 m para todas.

```
'RunOptions'

    RunOptions.NumTrialPoints= 5;
    RunOptions.NumStageOnePoints= 5;

    RunOptions.Algorithm= 'sqp';
    RunOptions.Display= 'off';
    RunOptions.TolFun= 10^(-2);
    RunOptions.TolX= 10^(-2);
    RunOptions.UseParallel= false;
```

Se trabaja con 5 puntos de inicio, con las tolerancias reflejadas (10^{-2}), y sin usar computación en paralelo.

```
    RunOptions.UsePrevData= 0;

    RunOptions.UseNNet= 1;
```

Como es la primera ejecución del problema, no se dispone de datos almacenados. Se ha decidido usar redes neuronales para realizar cálculos.

```
    RunOptions.DataPoolOrigin= 1;

    RunOptions.VolNNetnumNeurons= 10;
    RunOptions.ConstrNNetnumNeurons= 20;

    RunOptions.NNetTrainFcn= 'trainlm';
    RunOptions.NNetminInputs= 20;
    RunOptions.NNetnumLoop= 4;
    RunOptions.maxNNetUses= 5;
    RunOptions.maxFEMUses= 3;
```

Se definen parámetros relacionados con las redes neuronales:

- Se emplea la estrategia 1 a la hora de construir el conjunto de datos que se usarán para crear las redes neuronales (en el momento de la redacción de este manual sólo hay una implementada, luego el valor del campo *`DataPoolOrigin`* sólo puede ser 1).

- Se utilizan 10 neuronas para la construcción de las redes neuronales del volumen y 20 para las redes de las restricciones (en los campos *VolNNetnumNeurons* y *ConstrNNetnumNeurons* respectivamente).
- Se usa la función *trainlm* para el entrenamiento de las redes neuronales (en el campo *NNetTrainFcn*).
- Se requieren 20 datos provenientes de análisis MEF diferentes para la creación de las redes neuronales (en el campo *NNetminInputs*).
- Se crean 4 redes en bucle para el volumen y otras 4 para las restricciones, para la elección de las que tengan menor error de validación (en el campo *NnetnumLoop*).
- Unas redes neuronales existentes se pueden utilizar un máximo de 5 veces seguidas (en el campo *maxNNetUses*).
- El programa realiza 3 análisis MEF como máximo entre el fin de la utilización de unas redes existentes y la construcción de las siguientes (en el campo *maxFEMUses*).

```
RunOptions.NumConstr= 1;
```

Se tiene en cuenta sólo 1 restricción durante el proceso.

```
RunOptions.ScaledAxisXmin= 0;
RunOptions.ScaledAxisXmax= max(IniValRes.ub)+…
    min(IniValRes.lb);
RunOptions.ScaledAxisYmin= 0;
RunOptions.ScaledAxisYmax= max(IniValRes.ub)+…
    min(IniValRes.lb);
```

Se definen los ejes fijos que se utilizarán para representar las geometrías correspondientes.

```
'StartGeo'

    load('Geometria_NurbsOpt_4var_squared.mat');

    Geo.Cv(1).FixedCv=0;
    Geo.Cv(2).FixedCv=0;
    Geo.Cv(3).FixedCv=0;

    Geo.Material.MaxStress= 2e6;
    Geo.Material.ElastModul= 10100000;
    Geo.Material.StressFOS= 1;
```

Se carga el archivo (elaborado previamente en cgFEM) que contiene la geometría base del problema y los datos de las propiedades del material (*Geometria_NurbsOpt_4var_squared.mat*). Se indican las curvas de la geometría que no son fijas y se modifican algunos parámetros del material:

- Tensión máxima admisible ($2*10^6$ Pa).
- Módulo de elasticidad (10100000 Pa).
- Factor de seguridad de las tensiones (1).

```
'ProcessGeo'

    Geo.Pt.XYZ(2,4)=Reference.Parameters(4);
    Geo.Pt.XYZ(1,6)=Reference.Parameters(2);
    Geo.Pt.XYZ(2,6)=Reference.Parameters(3);
    Geo.Pt.XYZ(1,3)=Reference.Parameters(1);

    SensAnaGlob.Param.DesVarFile=...
        'sa_SetDesVar_NurbsOpt_4var.m';
```

Aquí es donde se asignan las coordenadas de determinados puntos de la geometría a las variables de diseño:

- La coordenada “y” del nodo 4 (punto 3 en la figura 30 y pico) es la variable 4.
- La coordenada “x” del nodo 6 (punto 2 en la figura 30 y pico) es la variable 2.
- La coordenada “y” del nodo 6 (punto 2 en la figura 30 y pico) es la variable 3.
- La coordenada “x” del nodo 3 (punto 1 en la figura 30 y pico) es la variable 1.

Se establece la función encargada de modificar la geometría con las variables de diseño (*sa_SetDesVar_NurbsOpt_4var.m*).

```
'MeshParam'

    BatchInfo.HadaptiveMethod=1;
    BatchInfo.AnalysisType='h_adaptive_mesh';
    BatchInfo.EType=2;
    BatchInfo.IniMesh=5;
    BatchInfo.IntsMesh=5;
    BatchInfo.MaxLevelMesh=11;
    BatchInfo.SquareMesh=1;
    BatchInfo.UniformRefinement=0;
    BatchInfo.RecovType=300;
    BatchInfo.TargErr=[-1 -1 -1 -1 1 -1];
    BatchInfo.ErrReduct=50;
    BatchInfo.MxRefInc=3;
    BatchInfo.MxRefLevel=11;
    BatchInfo.MxNIter=3;
    BatchInfo.GeometricalRefinamentArea=0;
    BatchInfo.GeometricalRefinamentCurvature=1;
    BatchInfo.ExactPressSolution=0;
    BatchInfo.AnalysisMode2D=PlaneStrain;
    BatchInfo.LoadCase2Solve=[1 2 3 4 5 6];
    BatchInfo.Error_Control=0;
    BatchInfo.ParTrans=Quadratic;
    BatchInfo.TerrorTerm=0;
    BatchInfo.MeshCoords=[-1.99 30;-1.99 30];
```

Se definen todos los parámetros necesarios para el proceso de mallado del bloque de análisis MEF:

- El tipo de elemento (en el campo *EType*) es cuadrático (2).
- El máximo error en norma energética deseado en la malla (en el campo *TargErr*) es 1%.
- El máximo número de iteraciones de la malla en el proceso h-adaptativo (en el campo *MxNIter*) es 3.
- Las coordenadas de la malla (en el campo *MeshCoords*) son [-1.99 30;-1.99 30].

Una vez se ha establecido el valor deseado de todos estos parámetros, se guarda el archivo y se procede a la ejecución del programa.

Lo que queda es un proceso muy simple; la intervención del usuario es mínima desde este punto hasta la presentación de resultados.

Se establece la carpeta maestra del programa como directorio de Matlab:

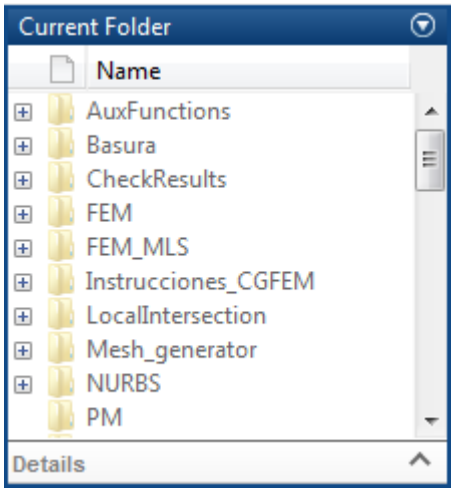

Figura 27. Directorio de Matlab al ejecutar el programa.

Se introduce el comando *GlobalOptimization*:

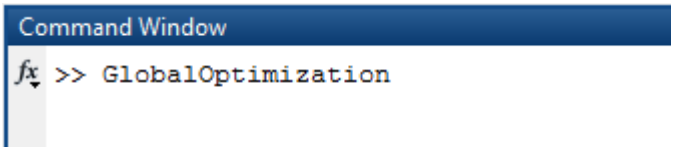

Figura 28. Introducción del comando *GlobalOptimization*.

A continuación, el programa abre una ventana de diálogo donde pide al usuario que seleccione con qué problema de la carpeta *samples* desea trabajar.

Si el usuario ha decidido usar datos almacenados, se abre inmediatamente otra ventana , donde el programa expone los archivos *.mat* de la carpeta *Stored ResultData* para que el usuario decida cuál utilizar.

Como ésta es la primera ejecución de este problema, no se dispone de dichos datos. Si el usuario hubiera editado `ProblemData.m` con `RunOptions.UsePrevData= 1`, la ventana habría mostrado una carpeta *Stored ResultData* vacía.

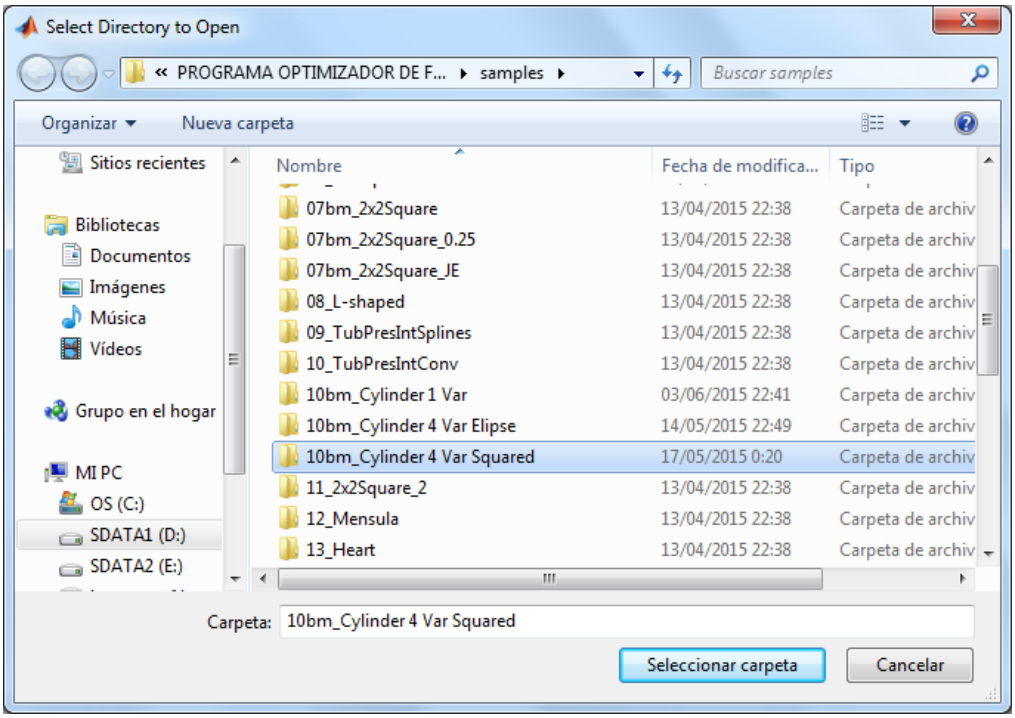

Figura 29. Ventana de selección de problema.

El programa continúa su ejecución hasta llegar a una solución, sin necesidad de más intervenciones del usuario.

Para acceder a los datos de la ejecución, el usuario debe ir a la carpeta del problema, donde encontrará dos subcarpetas: *Results* y *Stored ResultData*.

En la carpeta *Results* se almacenan los datos de todas las ejecuciones del problema. Para cada ejecución hay una carpeta con la fecha y hora exactas del inicio de ésta:

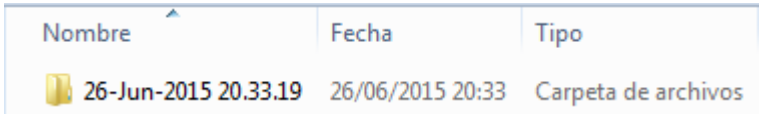

Figura 30. Contenido de la carpeta *Results*.

En la carpeta *Stored ResultData* se almacena el contenido de la variable `ResultData` de todas las ejecuciones del problema. Esta información está presente en la carpeta *Results* junto a otros muchos datos; la existencia de la carpeta *Stored ResultData* permite un acceso sencillo a datos de ejecuciones anteriores, y si el usuario quiere ahorrar espacio, o sólo le interesa el contenido de las variables `ResultData` para usar en análisis posteriores, puede eliminar los datos de la carpeta *Results* sin perder información útil para futuras ejecuciones del problema.

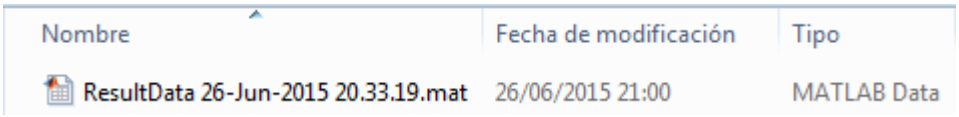

Figura 31. Contenido de la carpeta *Stored ResultData*.

Dentro de la carpeta de la ejecución que acaba de terminar se encuentran:

- Las subcarpetas que contienen las geometrías que cumplen las restricciones del proceso de optimización (en detalle y a la escala definida por el usuario).

- El archivo *Problem Data.mat*, donde está almacenado el contenido de las principales variables del problema.

- El archivo *Result Data.txt*, donde se escriben algunos parámetros de cada iteración del proceso de optimización:
  - Origen de la iteración.
  - Valor de la variable de diseño.
  - Valor del volumen.
  - Valor de la restricción.
  - Error en norma energética en la malla.
  - Tiempo de iteración.

- El archivo *Solution Data.txt*, donde se escriben los parámetros más significativos de todo el proceso de optimización:
  - Tiempo total de iteración (tiempo real que ha tardado la ejecución del problema).
  - Tiempo total de análisis (suma de los tiempos de análisis de cada iteración, aunque se hayan usado datos repetidos).
  - Número de iteraciones realizadas.
  - Número y procedencia de los análisis MEF realizados.
  - Número y procedencia de los cálculos con redes neuronales realizados (en este problema no se han utilizado).
  - Número y procedencia de las repeticiones de datos almacenados.
  - Número de la iteración en la que se ha encontrado la solución del problema.
  - Valor solución de la variable de diseño.
  - Valor solución del volumen.
  - Error en norma energética en la malla de la iteración solución.
  - Máximo error en norma energética entre todas las iteraciones, e iteración en la que se encuentra.

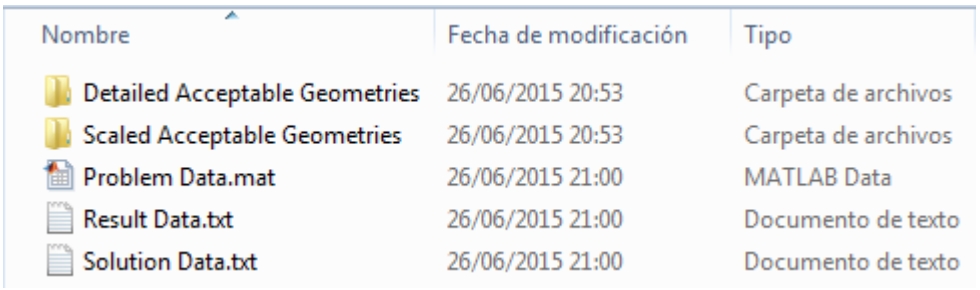

Figura 32. Contenido de la carpeta *26-Jun-2015 20.33.19*.

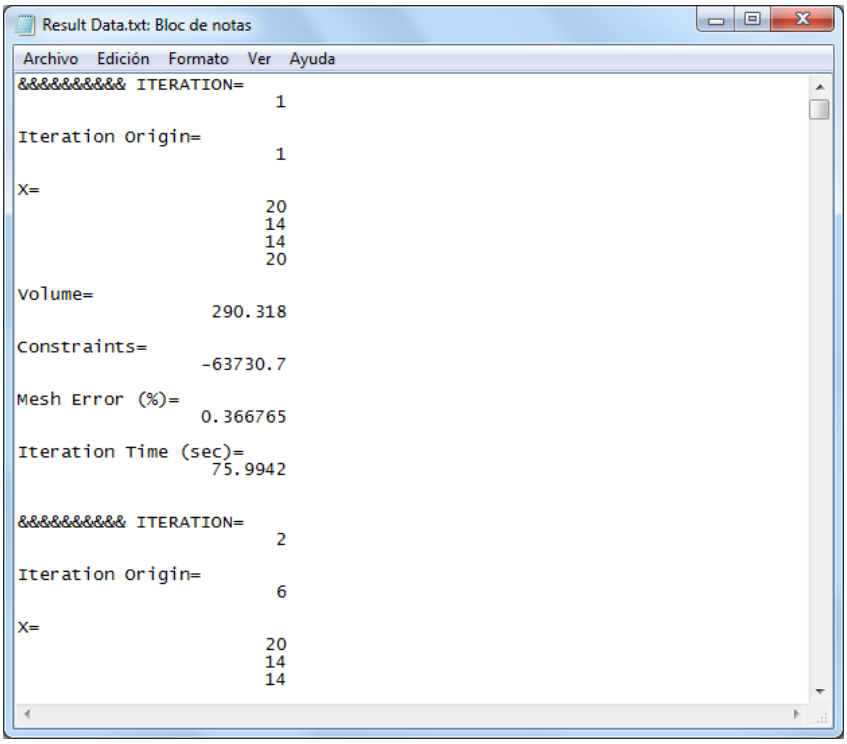

Figura 33. Contenido del archivo *Result Data.txt*.

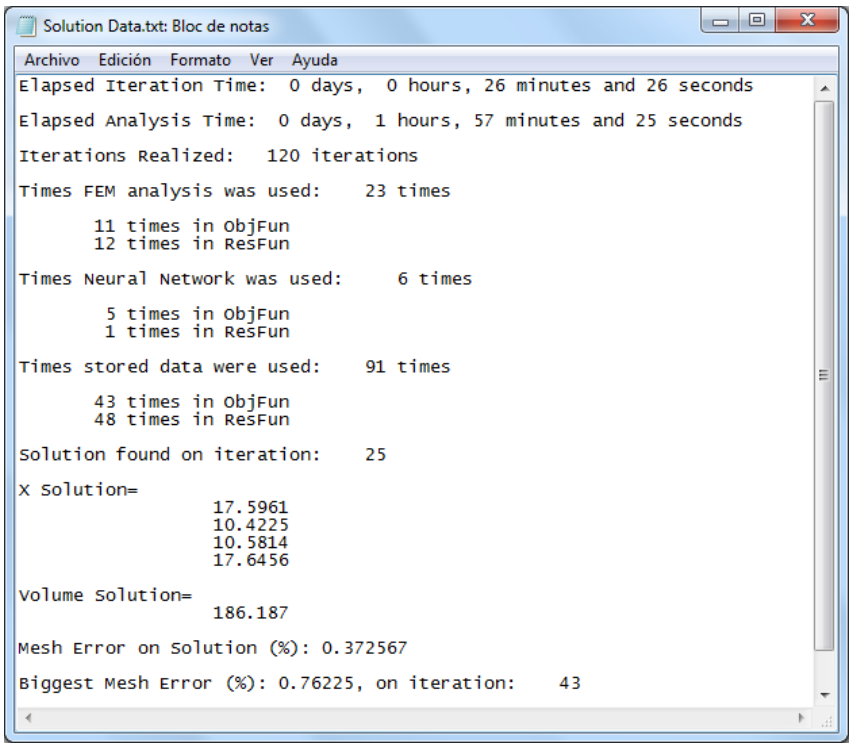

Figura 34. Contenido del archivo *Solution Data.txt*.

Si se produce algún error durante la ejecución de un problema, el programa crea la carpeta *Error Info*, con el siguiente contenido:

- Subcarpetas donde se almacenan las geometrías que dan lugar a errores (en detalle y a la escala definida por el usuario).
- El archivo *Error Data.txt*, donde se escribe la información del error.

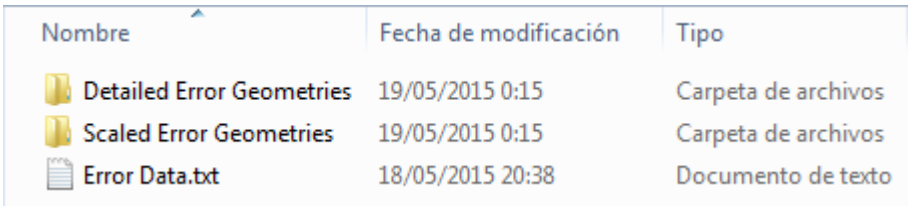

Figura 35. Ejemplo del contenido de una carpeta *Error Info*.

APÉNDICE B. MANUAL DEL PROGRAMADOR

En este apéndice se explicará con todo detalle el código del nuevo programa, tanto las principales variables que lo componen como los módulos y bloques en los que se estructura.

B.1. VARIABLES

Las variables descritas a continuación tienen carácter global, es decir, afectan a varios módulos del programa.

ProblemPath

Define el directorio del problema. El programa de optimización, nada más empezar su ejecución, le pide al usuario que seleccione qué problema desea ejecutar de entre los presentes en la carpeta *samples*. Almacena la dirección de éste para su uso inmediato.

ActualDate

Almacena, en una cadena de caracteres, la fecha y hora exactas de la ejecución en curso del programa.

ResultsPath

Define el directorio de la ejecución en curso del programa sirviéndose de la dirección almacenada en `ProblemPath` y la fecha y hora exactas almacenadas en `ActualDate`.

IniValRes

Define los campos de la función *createOptimProblem*: valores iniciales de las variables de diseño, propuestos por el usuario, así como sus límites y restricciones.

RunOptions

Esta variable es de gran importancia, ya que define una gran cantidad de datos relevantes:

- Las opciones de las funciones que definen el problema de optimización (*GlobalSearch*, *optimset*).
- La utilización de datos almacenados.
- El uso de redes neuronales, y multitud de parámetros de éstas.
- La numeración asignada al origen de los datos calculados.
- El número de restricciones del proceso de optimización.
- Los ejes para la representación de las geometrías calculadas a una escala determinada.

Geo

Define la geometría que se va a analizar: el contorno, las líneas que no son fijas (los puntos que las definen son las variables de diseño, que van cambiando a lo largo del proceso de optimización) y las propiedades del material.

Iteration

Indica el número de geometría considerada por el proceso de optimización. Su valor se inicializa en 0, y se incrementa en 1 cada vez que se analiza una geometría propuesta.

thereIsNNet

Indica al programa la existencia de redes neuronales. Su valor se inicializa en 0, y cambia a 1 cuando el programa las crea.

UseNNet

Indica al programa que debe usar las redes neuronales para hacer cálculos, en lugar de hacer un análisis MEF. Su valor se inicializa en 0, y cambia a 1 cuando corresponda, según los parámetros que el usuario haya definido en *RunOptions*.

numNNetUses

Indica al programa cuántas veces ha usado las redes neuronales para hacer cálculos. Su valor se inicializa en 0, y se incrementa en 1 cada vez que las utiliza.

numFEMUses

Indica al programa el número de análisis MEF que ha realizado tras terminar los cálculos con unas redes neuronales determinadas. Por tanto, esta variable sólo es relevante si el programa hace uso de las redes neuronales. Su valor se inicializa en 0, y se incrementa en 1 cada vez que realiza un análisis MEF bajo esa condición.

SensAnaIter

Contiene los resultados del problema de sensibilidades por iteración. Se inicializa al comienzo de cada análisis MEF, vaciándose.

SensAnaGlob

Contiene los resultados del problema de sensibilidades global. Se inicializa al comienzo de cada análisis MEF.

Reference

Contiene algunos resultados de los análisis MEF; los valores de:

- La derivada del volumen por elemento.

- La derivada del volumen de la geometría.
- La tensión de Von Mises por elemento.
- La derivada de la tensión de Von Mises por elemento.
- La tensión máxima de Von Mises.
- La derivada de la tensión máxima de Von Mises.

BatchInfo

Contiene numerosos parámetros necesarios para el proceso de mallado del análisis MEF, siendo los más significativos el tipo de elemento (lineal o cuadrático), el error máximo en norma energética deseado, y el número máximo de iteraciones de malla del proceso h-adaptativo.

bestnetVol

Contiene la mejor red neuronal creada para calcular el volumen y sus derivadas.

bestnetConstr

Contiene la mejor red neuronal creada para calcular las restricciones y sus derivadas.

normx0FEMUsefulData

Contiene las normas de las variables de diseño de todas las geometrías cuyos datos han sido usados para crear la red neuronal usada por el programa de optimización.

ResultData

Es la variable más importante del programa. Almacena por columnas datos significativos de cada iteración del proceso de optimización:

- Origen de la iteración.
- Origen de los datos.

- Variables de diseño.
- Volumen.
- Derivadas del volumen.
- Restricciones.
- Derivadas de las restricciones.
- Número de malla del proceso h-adaptativo.
- Error en norma energética en dicha malla.
- Número de grados de libertad.
- Número de elementos.
- Tiempo de análisis.
- Tiempo de iteración.

PrevResData

Esta variable sólo es relevante en los análisis de un problema determinado en los que se disponga de datos calculados en algún análisis anterior de ese mismo problema (y se decida su uso). Es el *ResultData* de dicho análisis.

SolutionIt

Es la iteración del programa de optimización en la que se encuentra la solución; donde se consigue el objetivo final: hallar el mínimo volumen que soporta unas cargas determinadas cumpliendo las restricciones requeridas.

GlobOptIsDone

Indica al programa si el proceso de optimización ha terminado. El uso de esta variable es marginal: si el programa trabajase con redes neuronales, existiría la posibilidad de que el resultado final (volumen) fuera calculado mediante redes neuronales, y no mediante un análisis MEF. En ese caso, el programa realizaría un análisis MEF de la geometría solución propuesta, y sustituiría en la variable *ResultData*, en la iteración correspondiente, los datos hallados con redes neuronales por los calculados mediante el análisis MEF, incluidos el número de malla y el error en norma energética, los cuales se

calculan en la rutina `ErrorCalc.m`. Dicha rutina necesita que se le indique que debe guardar los datos mencionados en la iteración de la solución, no en la última iteración del programa.

La existencia de esta variable es, pues, consecuencia de la forma en la que está escrito el código. Su valor se inicializa en 0 y cambia a 1 una vez el proceso de optimización ha finalizado y, por tanto, la solución ha sido hallada.

B.2. MÓDULOS

El programa desarrollado es vasto. La consecución de una comprensión del mismo lo más sencilla posible, tanto para el usuario como el programador, y el propio lenguaje de programación de Matlab, hacen necesaria la división del programa en varias partes o módulos:

- `GlobalOptimization.m`.
- `ObjectiveFunction.m`.
- `RestrictionFunction.m`.
- `CreateNNetDataPool.m`.
- `ProblemData.m`.

A su vez, cada módulo está formado por bloques, cada uno con un propósito, fácilmente identificables y desarrollados a continuación.

B.2.1. *GlobalOptimization.m*

Es el módulo central del programa; la ejecución de un problema empieza siempre con la llamada a esta función. Está presente en el directorio principal del programa.

Contiene los comandos que definen e inician el proceso de optimización, así como instrucciones para poner a disposición del usuario los datos más importantes de la ejecución realizada y almacenarlos en un archivo *.mat*.

A continuación se presenta y explica el código en detalle:

```
function GlobalOptimization

clear all;

restoredefaultpath;

global ProblemPath ResultsPath IniValRes RunOptions Iteration...
    ResultData Geo FEMGui BatchInfo ActualDate PrevResData...
    thereIsNNet UseNNet numNNetUses numFEMUses GlobOptIsDone...
    SolutionIt Param Nod Elm Errores Iter iMesh CalcMesh...
    GlobalMatKData GlobalIntersData SensAnaGlob SensAnaIter...
    MatK Reference
```

Se define la función *GlobalOptimization.m*, se eliminan todas las variables, se inicializan los directorios donde trabaja Matlab, y se declaran una serie de variables globales que serán necesarias para el buen funcionamiento del programa.

```
ProblemPath = uigetdir(fullfile(pwd,'samples'));

Paths;
addpath(ProblemPath);

Constants;
```

El programa pide al usuario que seleccione qué problema de la carpeta *samples* quiere ejecutar. Se añaden al *Path* de Matlab todos los directorios necesarios para el funcionamiento del calculador (necesario para los análisis MEF) mediante la instrucción *Paths*, y el directorio del problema seleccionado. Se declara otra serie de variables globales, necesarias para el funcionamiento del calculador, mediante la instrucción *Constants*.

```
if ~exist(fullfile(ProblemPath,'Results'),'dir')
mkdir(fullfile(ProblemPath,'Results'));
end

if ~exist(fullfile(ProblemPath,'Stored ResultData'),'dir')
mkdir(fullfile(ProblemPath,'Stored ResultData'));
end
```

En el directorio del problema seleccionado, se crean (si no existen ya) dos carpetas: *Results*, donde se almacenarán las carpetas de todas las ejecuciones del problema, y *Stored ResultData*, donde se guardarán los archivos *`ResultData.mat`* de todas las ejecuciones del problema, para su potencial uso en ejecuciones posteriores.

```
ActualDate= strrep(datestr(clock), ':' , '.' );
ResultsPath=fullfile(ProblemPath,'Results',ActualDate);

if ~exist(ResultsPath,'dir')
mkdir(ResultsPath);
end

if ~exist(fullfile(ResultsPath,'Detailed Acceptable...
    Geometries'),'dir')
mkdir(fullfile(ResultsPath,'Detailed Acceptable Geometries'));
end

if ~exist(fullfile(ResultsPath,'Scaled Acceptable...
    Geometries'),'dir')
mkdir(fullfile(ResultsPath,'Scaled Acceptable Geometries'));
end
```

Usando la fecha y hora exactas de la ejecución en curso, el programa crea la carpeta que será el directorio de la actual ejecución del programa, así como las subcarpetas donde se almacenarán las geometrías que cumplan las restricciones impuestas.

El código hasta este punto se puede interpretar como el “bloque de preparación” de *`GlobalOptimization.m`*, pues en él se definen los directorios donde el programa obtendrá y almacenará sus datos.

```
ProblemData('IniValRes');
ProblemData('RunOptions');
RunOptions.ObjFunFEM= 1;
RunOptions.ObjFunNNet= 2;
RunOptions.ObjFunRepeat= 3;
RunOptions.ResFunFEM= 4;
RunOptions.ResFunNNet= 5;
RunOptions.ResFunRepeat= 6;
ProblemData('StartGeo');
```

Se cargan los datos definidos por el usuario en *ProblemData.m* para la ejecución en curso. Se establecen los valores que definen el origen de los datos; en principio no hay motivos para cambiar los ya existentes, ni es recomendable, pues el programa decide qué datos almacenados utilizar en los distintos módulos en base a estos valores.

```
if RunOptions.UsePrevData== 1
    try
        FileName=uigetfile(fullfile(ProblemPath,'Stored...
            ResultData'),'*.mat');
        LoadedData=load(fullfile(ProblemPath,'Stored...
            ResultData',FileName));
        PrevResData= LoadedData.ResultData;
    catch
        fprintf('NO FILE SELECTED\r\n')
        RunOptions.UsePrevData= 0;
    end
end
```

Si el usuario elige la opción de usar datos almacenados, el programa le pregunta qué archivo *.mat* de la carpeta *Stored ResultData* desea utilizar, y carga los datos pertinentes en la variable *PrevResData*. Si, en el último momento, el usuario cambia de idea y no selecciona ningún archivo, el programa se lo hace saber y continúa su ejecución de forma normal.

```
Iteration= 0;
thereIsNNet= 0;
UseNNet= 0;
numNNetUses= 0;
numFEMUses= 0;
GlobOptIsDone= 0;
```

Se inicializan estas variables, necesarias para el funcionamiento del programa.

```
gs= GlobalSearch('NumTrialPoints',RunOptions.NumTrialPoints,...
    'NumStageOnePoints',RunOptions.NumStageOnePoints,...
    'StartPointsToRun','bounds-ineqs');
```

Se construye *GlobalSearch* con los datos definidos por el usuario en *ProblemData.m*.

```
opts= optimset('Algorithm',RunOptions.Algorithm,...
    'Display', RunOptions.Display,...
    'GradConstr','on',...
    'GradObj','on',...
    'TolFun',RunOptions.TolFun,...
    'TolX',RunOptions.TolX,...
    'UseParallel',RunOptions.UseParallel);

problem= createOptimProblem('fmincon',...
    'objective',@ObjectiveFunction,...
    'x0',IniValRes.x0,...
    'Aineq',IniValRes.Aineq,...
    'bineq',IniValRes.bineq,...
    'Aeq',IniValRes.Aeq,...
    'beq',IniValRes.beq,...
    'lb',IniValRes.lb,...
    'ub',IniValRes.ub,...
    'nonlcon',@RestrictionFunction,...
    'options',opts);
```

Se construye el problema de optimización con los datos definidos por el usuario en *ProblemData.m*.

```
[xmin,fmin,~,~,~] = run(gs,problem);
```

Éste es el comando que inicia el problema de optimización. Sólo importan dos salidas: los valores solución de las variables de diseño (*xmin*) y el valor solución del volumen (*fmin*).

Esta nueva sección del programa se puede interpretar como el “bloque de desarrollo” de *GlobalOptimization.m*, pues en ella se crea e inicia el problema de optimización.

```
GlobOptIsDone= 1;
```

Se indica a la rutina *ErrorCalc.m* que el proceso de optimización ha terminado.

```
Check= ResultData(3:length(xmin)+2,:)-xmin'*...
    ones(1,size(ResultData,2));
for i=1:size(ResultData,2)
    if norm(Check(1:length(xmin),i))<= 10*eps
        SolutionIt= i;
        break
    end
end
```

El programa busca en la variable *ResultData* en qué iteración se ha llegado a la solución del problema de optimización, almacenando el número de dicha iteración en la variable *SolutionIt*.

```
SolutionDataOrigin=ResultData(2,SolutionIt);
```

A continuación, el programa procede a identificar el origen de los datos solución (pueden haber sido hallados mediante análisis MEF o cálculos con redes neuronales).

Importante: si los datos se han obtenido mediante el uso de redes neuronales, el programa procede a la realización de un análisis MEF para que el usuario compare resultados. El código que se expone a continuación corresponde al "bloque de análisis MEF" del módulo *ObjectiveFunction.m* modificado para que se almacenen los resultados en la iteración solución *SolutionIt*, sobreescribiendo a los hallados mediante redes neuronales.

Para una explicación detallada del "bloque de análisis MEF", consultar el apartado dedicado al módulo *ObjectiveFunction.m*.

```
if SolutionDataOrigin==RunOptions.ObjFunNNet ||…
    SolutionDataOrigin==RunOptions.ResFunNNet

    NNetVolSol= fmin;

    tStart= tic;

    IterOrigin= RunOptions.ObjFunFEM;
    DataOrigin= IterOrigin;

    ResultData(1,SolutionIt)= IterOrigin;
    ResultData(2,SolutionIt)= DataOrigin;
```

```
ResultData(3:length(xmin)+2,SolutionIt)= xmin';
Exception= 0;

try

    Param.ModeBatch=1;

    Param.Contour=1;

    ProblemData('DesVarIni');
    IniConstants;

    Reference.Parameters=xmin;
    ProblemData('ProcessGeo');
    run(SensAnaGlob.Param.DesVarFile);
    ReCalcAll;

    ProblemData('MeshParam');
    if BatchInfo.SquareMesh
        dummymin=min(BatchInfo.MeshCoords(:,1));
        dummymax=max(BatchInfo.MeshCoords(:,2));
        BatchInfo.MeshCoords=[dummymin dummymax;dummymin...
            dummymax];
    end

    close all;

    DGeo(0,0,0);

    saveas(gcf,fullfile(ResultsPath,'Detailed Acceptable...
        Geometries', strcat(['Geometry' ' '...
            num2str(Iteration) ' Detailed (Solution)'...
                '.jpg'])));

    axis([RunOptions.ScaledAxisXmin...
        RunOptions.ScaledAxisXmax...
            RunOptions.ScaledAxisYmin...
                RunOptions.ScaledAxisYmax]);
    saveas(gcf,fullfile(ResultsPath,'Scaled Acceptable...
        Geometries', strcat(['Geometry' ' '...
            num2str(Iteration) ' Scaled (Solution)'...
                '.jpg'])));

    Source=2;
    MainGE(Source)

    sa_deltaV(iMesh,0);

    ResultData(length(xmin)+3,SolutionIt)=...
        Iter(iMesh).Volume.Vtotal;
    ResultData(length(xmin)+4:2*length(xmin)+3,SolutionIt)=...
        Reference.dV';
```

```
        sa_StressVM(iMesh,0);

        c=[Reference.MaxStressVM*Geo.Material.StressFOS-...
            Geo.Material.MaxStress, ];
        gradc=[Reference.dStressOfMaxStressVM, ];

        ResultData(2*length(xmin)+4:2*length(xmin)+3+...
            RunOptions.NumConstr,SolutionIt)= c';
        ResultData(2*length(xmin)+4+RunOptions.NumConstr:...
            (length(xmin)+1)*(RunOptions.NumConstr+2)+1,...
                SolutionIt)= gradc';

        ResultData((length(xmin)+1)*(RunOptions.NumConstr+2)+4,...
            SolutionIt)= CalcMesh(iMesh).NNodes*2;
        ResultData((length(xmin)+1)*(RunOptions.NumConstr+2)+5,...
            SolutionIt)= CalcMesh(iMesh).NumEle;

        tEndAna= toc(tStart);
        tEndIter= tEndAna;

        ResultData((length(xmin)+1)*(RunOptions.NumConstr+2)+6,...
            SolutionIt)= tEndAna;
        ResultData((length(xmin)+1)*(RunOptions.NumConstr+2)+7,...
            SolutionIt)= tEndIter;

    catch ME

        Exception= 1;

        if ~exist(fullfile(ResultsPath,'Error Info'),'dir')
        mkdir(fullfile(ResultsPath,'Error Info'));
        end

        if ~exist(fullfile(ResultsPath,'Error Info','Detailed...
            Error Geometries'),'dir')
        mkdir(fullfile(ResultsPath,'Error Info','Detailed...
            Error Geometries'));
        end

        if ~exist(fullfile(ResultsPath,'Error Info','Scaled...
            Error Geometries'),'dir')
        mkdir(fullfile(ResultsPath,'Error Info','Scaled...
            Error Geometries'));
        end

        close all;

        DGeo(0,0,0);

        saveas(gcf,fullfile(ResultsPath,'Error Info','Detailed...
            Error Geometries', strcat(['Geometry' ' '...
```

```
            num2str(SolutionIt) ' Detailed (Solution)'...
                '.jpg'])));

    axis([RunOptions.ScaledAxisXmin...
        RunOptions.ScaledAxisXmax...
            RunOptions.ScaledAxisYmin...
                RunOptions.ScaledAxisYmax]);
    saveas(gcf,fullfile(ResultsPath,'Error Info','Scaled...
        Error Geometries', strcat(['Geometry' ' '...
            num2str(SolutionIt) ' Scaled (Solution)'...
                '.jpg'])));

    Error_output = fullfile(ResultsPath,'Error Info','Error...
        Data.txt');
    fid = fopen(Error_output,'a');
    fprintf(fid,'&&&&&&&&&& ITERATION OF SOLUTION...
        (%5G)\r\n',SolutionIt);
    fprintf(fid,'%s\r\n',ME.identifier);
    fprintf(fid,'%s\r\n\r\n',ME.message);
    fprintf(fid,'FAILURES:\r\n');


        for e=1:length(ME.stack)
            fprintf(fid,'%s at %i\r\n',...
                ME.stack(e).name,ME.stack(e).line);
        end
    fprintf(fid,'\r\n');
    fprintf(fid,'X value=\r\n');
    fprintf(fid,'%25G\r\n', xmin');
    fprintf(fid,'\r\n\r\n');
    fclose(fid);

    ResultData(length(xmin)+3,SolutionIt)= nan;
    ResultData(length(xmin)+4:2*length(xmin)+3,SolutionIt)=...
        nan(1,length(xmin))';

    ResultData(2*length(xmin)+4:2*length(xmin)+3+...
        RunOptions.NumConstr,SolutionIt)=...
            [Geo.Material.MaxStress, ]';
    ResultData(2*length(xmin)+4+RunOptions.NumConstr:...
        (length(xmin)+1)*(RunOptions.NumConstr+2)+1,...
            SolutionIt)= nan(1,RunOptions.NumConstr*...
                length(xmin))';

    ResultData((length(xmin)+1)*(RunOptions.NumConstr+2)+2,...
        SolutionIt)= nan;
    ResultData((length(xmin)+1)*(RunOptions.NumConstr+2)+3,...
        SolutionIt)= nan;

    ResultData((length(xmin)+1)*(RunOptions.NumConstr+2)+4,...
        SolutionIt)= nan;
    ResultData((length(xmin)+1)*(RunOptions.NumConstr+2)+5,...
```

```
            SolutionIt)= nan;

        tEndAna= toc(tStart);
        tEndIter= tEndAna;

        ResultData((length(xmin)+1)*(RunOptions.NumConstr+2)+6,...
            SolutionIt)=tEndAna;
        ResultData((length(xmin)+1)*(RunOptions.NumConstr+2)+7,...
            SolutionIt)=tEndIter;

    end

end
```

Aquí acaba el “bloque de análisis MEF”.

Recordatorio: consultar el apartado dedicado al módulo *`ObjectiveFunction.m`* para una explicación detallada.

```
MeshErrorVector= ResultData((length(xmin)+1)*...
    (RunOptions.NumConstr+2)+3,:);
BME= max(MeshErrorVector);
BMEIter= find(MeshErrorVector==BME, 1);
```

Se extrae de la variable *`ResultData`* el máximo error en norma energética en la malla de entre todas las iteraciones.

```
IterOriginVector= ResultData(1,:);

TimesObjFunFEM= length(find(IterOriginVector==...
    RunOptions.ObjFunFEM));
TimesResFunFEM= length(find(IterOriginVector==...
    RunOptions.ResFunFEM));
TimesFEM= TimesObjFunFEM + TimesResFunFEM;

TimesObjFunNNet= length(find(IterOriginVector==...
    RunOptions.ObjFunNNet));
TimesResFunNNet= length(find(IterOriginVector==...
    RunOptions.ResFunNNet));
TimesNNet= TimesObjFunNNet + TimesResFunNNet;

TimesObjFunRepeat= length(find(IterOriginVector==...
    RunOptions.ObjFunRepeat));
TimesResFunRepeat= length(find(IterOriginVector==...
```

```
    RunOptions.ResFunRepeat));
TimesRepeat= TimesObjFunRepeat + TimesResFunRepeat;
```

Se extrae de la variable *ResultData* el número de veces que los datos han sido almacenados en ésta mediante un análisis MEF, el uso de las redes neuronales, o la repetición de resultados, así como el módulo del que provienen (*ObjectiveFunction.m* o *RestrictionFunction.m*).

```
tAnaTot= sum(ResultData((length(xmin)+1)*...
    (RunOptions.NumConstr+2)+6,:));

AnaSec= floor(rem(tAnaTot,60));
AnaMin= rem(floor(tAnaTot/60),60);
AnaHours= rem(floor(floor(tAnaTot/60)/60),24);
AnaDays= floor(floor(floor(tAnaTot/60)/60)/24);
```

Se extrae de la variable *ResultData* el tiempo total de análisis en formato días/horas/minutos/segundos.

```
tIterTot= sum(ResultData((length(xmin)+1)*...
    (RunOptions.NumConstr+2)+7,:));

IterSec= floor(rem(tIterTot,60));
IterMin= rem(floor(tIterTot/60),60);
IterHours= rem(floor(floor(tIterTot/60)/60),24);
IterDays= floor(floor(floor(tIterTot/60)/60)/24);
```

Se extrae de la variable *ResultData* el tiempo total de iteración en formato días/horas/minutos/segundos.

```
Solution_output = fullfile(ResultsPath,'Solution Data.txt');
```

Se crea el archivo *Solution Data.txt*, que contendrá toda la información útil de la ejecución del problema.

```
fid = fopen(Solution_output,'a');
fprintf(fid,'Elapsed Iteration Time: %2G days, %2G hours,...
    %2G minutes and %2G seconds\r\n\r\n',...
        IterDays,IterHours,IterMin,IterSec);
fprintf(fid,'Elapsed Analysis Time: %2G days, %2G hours,...
    %2G minutes and %2G seconds\r\n\r\n',...
        AnaDays,AnaHours,AnaMin,AnaSec);
fprintf(fid,'Iterations Realized: %5G iterations\r\n\r\n',...
    Iteration);
fprintf(fid,'Times FEM analysis was used: %5G times\r\n\r\n',...
    TimesFEM);
fprintf(fid,'%5G times in ObjFun\r\n',TimesObjFunFEM);
fprintf(fid,'%5G times in ResFun\r\n\r\n',TimesResFunFEM);
fprintf(fid,'Times Neural Network was used: %5G times\r\n\r\n',...
    TimesNNet);
fprintf(fid,'%5G times in ObjFun\r\n',TimesObjFunNNet);
fprintf(fid,'%5G times in ResFun\r\n\r\n',TimesResFunNNet);
fprintf(fid,'Times stored data were used: %5G times\r\n\r\n',...
    TimesRepeat);
fprintf(fid,'%5G times in ObjFun\r\n',TimesObjFunRepeat);
fprintf(fid,'%5G times in ResFun\r\n\r\n',TimesResFunRepeat);
```

Se escribe en *Solution Data.txt* los tiempos de iteración y de análisis, el número de iteraciones realizadas y el número de veces que los datos han sido almacenados en *`ResultData`* mediante un análisis MEF, el uso de las redes neuronales, o la repetición de resultados, así como el módulo del que provienen (*`ObjectiveFunction.m`* o *`RestrictionFunction.m`*).

```
fprintf(fid,'Solution found on iteration: %5G\r\n\r\n',...
    SolutionIt);
fprintf(fid,'X Solution=\r\n');
fprintf(fid,'%25G\r\n',xmin');
fprintf(fid,'\r\n');
```

Se escribe en *Solution Data.txt* el número de iteración donde se ha alcanzado la solución, con el correspondiente valor de las variables de diseño.

```
if exist('NNetVolSol','var')

    fprintf(fid,'Volume Solution (NNet)=\r\n');
    fprintf(fid,'%25G\r\n\r\n',NNetVolSol);

    if Exception==0
```

```
        fprintf(fid,'Volume Solution (FEM)=\r\n');
        fprintf(fid,'%25G\r\n\r\n',ResultData((length(xmin)+3),...
            SolutionIt));
        fprintf(fid,'C on Solution (FEM)=\r\n');
        fprintf(fid,'%25G\r\n',ResultData(2*length(xmin)+4:...
            2*length(xmin)+3+RunOptions.NumConstr,...
                SolutionIt));
        fprintf(fid,'\r\n');
        fprintf(fid,'Mesh Error on Solution (%%): %5G\r\n\r\n',...
            ResultData((length(xmin)+1)*(RunOptions.NumConstr+...
                2)+3,SolutionIt));
    else
        fprintf(fid,'Volume Solution (FEM) was not calculated...
            because of an error\r\n\r\n');
    end

else

    fprintf(fid,'Volume Solution=\r\n');
    fprintf(fid,'%25G\r\n\r\n',ResultData((length(xmin)+3),...
        SolutionIt));
    fprintf(fid,'Mesh Error on Solution (%%): %5G\r\n\r\n',...
        ResultData((length(xmin)+1)*(RunOptions.NumConstr+...
            2)+3,SolutionIt));

end
```

Se escribe en *Solution Data.txt* el volumen solución y el error en norma energética en la malla correspondiente.

Si la solución original ha sido hallada mediante redes neuronales, el usuario puede comparar su valor con el obtenido en el análisis MEF que ha tenido lugar, pues se escriben ambas. También puede valorar su validez real, pues se escribe el valor de las restricciones (la solución es válida si las restricciones < 0). Por último, si ha ocurrido algún error en el análisis MEF, se pone en conocimiento del usuario.

```
fprintf(fid,'Biggest Mesh Error (%%): %5G, on iteration:...
    %5G\r\n\r\n', BME, BMEIter);
fclose(fid);
```

Se escribe en *Solution Data.txt* el máximo error en norma energética en la malla de entre todas las iteraciones, por si fuera de interés para el usuario.

```
if exist('NNetVolSol','var')
    save(fullfile(ProblemPath,'Stored ResultData',...
        strcat(['ResultData' ' ' num2str(ActualDate)...
            '.mat'])),'ResultData');
end

save(fullfile(ResultsPath,'Problem Data.mat'));

fprintf('&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&\r\n')
fprintf('&&&&&&&&&&&&&&& PROBLEM SOLVED &&&&&&&&&&&&&&&\r\n')
fprintf('&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&\r\n')
```

Si la solución original ha sido hallada mediante redes neuronales, el programa guarda una última vez en la carpeta *Stored ResultData* los datos almacenados en la variable *`ResultData`*, ya que los datos en la iteración de la solución han sido sobreescritos.

Por último, se guarda en el directorio de la ejecución en curso el archivo *`Problem Data.mat`*, que contiene todos los datos útiles de dicha ejecución.

Esta última sección puede considerarse el “bloque de presentación de resultados”.

B.2.2. *ObjectiveFunction.m*

Es uno de los dos módulos que componen el proceso de optimización. Está presente en el directorio principal del programa.

Se encarga de calcular el valor de la función que se pretende optimizar (volumen) y sus derivadas para unos valores determinados de las variables de diseño.

A continuación se presenta y explica el código en detalle:

```
function [Volume,GradVolume]= ObjectiveFunction(x0)

global Geo Param FEMGui Nod Elm Errores BatchInfo Iter iMesh...
    CalcMesh GlobalMatKData GlobalIntersData SensAnaGlob...
    SensAnaIter MatK Reference ResultsPath RunOptions Iteration...
    ResultData PrevResData ProblemPath ActualDate...
    thereIsNNet UseNNet numNNetUses numFEMUses...
    normx0FEMUsefulData bestnetVol bestnetConstr
```

```
Constants;
Iteration= Iteration+1;
```

Se define la función *ObjectiveFunction.m*, se declara una serie de variables globales que serán necesarias para el buen funcionamiento del programa y se incrementa en 1 el número de iteración.

```
Repeat= 0;

if RunOptions.UsePrevData==1 && Iteration>1
    RepeatDataPool= [PrevResData,ResultData(:,1:Iteration-1)];
elseif RunOptions.UsePrevData==1 && Iteration==1
    RepeatDataPool= PrevResData;
elseif Iteration>1
    RepeatDataPool= ResultData(:,1:Iteration-1);
else
    RepeatDataPool= [ ];
end
```

En principio, los valores de las variables de diseño sugeridos por el programa de optimización no se corresponden con datos almacenados.

La variable *RepeatDataPool* contiene todos los datos que el programa tiene en cuenta a la hora de detectar la repetición de los valores de las variables de diseño sugeridos por el programa de optimización. Para definirla, se consideran todas las situaciones posibles, se usen datos de ejecuciones anteriores o no.

```
if ~isempty(RepeatDataPool)

    Check= RepeatDataPool(3:length(x0)+2,:)-x0'*...
        ones(1,size(RepeatDataPool,2));
    for i=1:size(RepeatDataPool,2)
        if norm(Check(1:length(x0),i))<=RunOptions.TolX*norm(x0)
            Repeat= 1;
            RepeatIt= i;
            break
        end
    end

end
```

Si *RepeatDataPool* tiene datos entre los que buscar, el programa intenta encontrar una coincidencia entre los valores de las variables de diseño almacenadas y los sugeridos por el programa de optimización.

```
if RunOptions.UseNNet==1 && Repeat==0

    NNetDataPool= CreateNNetDataPool(RunOptions.DataPoolOrigin);

    if ~isempty(NNetDataPool)
        originVector= NNetDataPool(2,:);
        FEMDataPos= find(originVector==RunOptions.ObjFunFEM|...
            originVector==RunOptions.ResFunFEM);
        FEMDataMatrix= NNetDataPool(3:(length(x0)+1)*...
            (RunOptions.NumConstr+2)+1,FEMDataPos);
        FEMUsefulData= unique(FEMDataMatrix','rows')';

        if thereIsNNet==0 && size(FEMUsefulData,2)>=...
            RunOptions.NNetminInputs
            UseNNet= 1;
        elseif thereIsNNet==1 && (norm(x0)<...
            min(normx0FEMUsefulData)||norm(x0)>...
                max(normx0FEMUsefulData))
            UseNNet= 0;
            thereIsNNet= 0;
            numNNetUses= 0;
        end
    end

end
```

Si el usuario ha decidido usar redes neuronales y los valores de las variables de diseño propuestas por el programa de optimización no son repetidos, se crea la variable *NNetDataPool*, cuyo contenido será utilizado para crear las redes neuronales. Dicho contenido es definido en el módulo *CreateNNetDataPool.m*, permitiendo tantas estrategias como se desee.

Si *NNetDataPool* tiene datos con los que trabajar, el programa contabiliza cuántos son útiles para construir las redes neuronales (datos provenientes del análisis MEF de distintas variables de diseño) y los almacena en la variable *FEMUsefulData*. Si hay datos suficientes (que sean suficientes depende de lo que el usuario haya definido en el módulo *ProblemData.m*), el programa procede a la construcción de las redes neuronales para hacer los cálculos pertinentes.

Si el programa está trabajando con unas redes neuronales determinadas y el proceso de optimización sugiere unos valores de las variables de diseño que se sitúan fuera del rango de datos usados para entrenar dichas redes neuronales, se procede al análisis MEF de los nuevos valores de las variables de diseño y las redes neuronales existentes son descartadas.

Esta primera sección del módulo *ObjectiveFunction.m* se puede considerar su "bloque de preparación".

Importante: a continuación se presentan los tres bloques principales del módulo *ObjectiveFunction.m*, que se corresponden con las tres formas posibles de obtener los valores de salida de éste (el volumen y sus derivadas): la repetición de resultados, el uso de redes neuronales y el análisis MEF.

```
if Repeat==1

    tStartIter= tic;

    IterOrigin= RunOptions.ObjFunRepeat;
    ResultData(1,Iteration)= IterOrigin;

    ResultData(2:(length(x0)+1)*(RunOptions.NumConstr+2)+6,...
        Iteration)=RepeatDataPool(2:(length(x0)+1)*...
            (RunOptions.NumConstr+2)+6,RepeatIt);

    Volume= ResultData(length(x0)+3,Iteration);
    GradVolume= ResultData(length(x0)+4:2*length(x0)+3,...
        Iteration)';

    tEndIter= toc(tStartIter);
    ResultData((length(x0)+1)*(RunOptions.NumConstr+2)+7,...
        Iteration)= tEndIter;
```

Si es posible usar datos almacenados, toda la información necesaria para dar salida a *ObjectiveFunction.m* (el volumen y sus derivadas) está inmediatamente a disposición del programa, por lo que éste escribe los datos repetidos en la columna de la variable *ResultData* correspondiente a la presente iteración.

Los únicos datos que no se repiten son el tiempo de iteración y el origen de la iteración.

Éste es el "bloque de repetición" del módulo *ObjectiveFunction.m*.

```
elseif UseNNet==1

    tStart= tic;

    numNNetUses=numNNetUses+1;

    IterOrigin= RunOptions.ObjFunNNet;
    DataOrigin= IterOrigin;

    ResultData(1,Iteration)= IterOrigin;
    ResultData(2,Iteration)= DataOrigin;
    ResultData(3:length(x0)+2,Iteration)= x0';
```

Si no es posible usar datos almacenados pero sí utilizar redes neuronales, el programa entra en este bloque y el contador del número de usos se incrementa en 1.

El origen de la iteración y el origen de los datos coinciden, y su valor es definido por el usuario en el módulo *ProblemData.m*.

Cuando el programa desee utilizar redes neuronales, es posible que éstas estén definidas, o no.

```
    if thereIsNNet==1
        OutputVol= bestnetVol(x0);
        Volume= OutputVol(1,1);
        GradVolume= OutputVol(2:length(x0)+1,1)';

        ResultData(length(x0)+3,Iteration)= Volume;
        ResultData(length(x0)+4:2*length(x0)+3,Iteration)=...
            GradVolume';
```

Si las redes neuronales han sido creadas en iteraciones anteriores, el volumen y sus derivadas se obtienen fácilmente, y se almacenan en la variable *ResultData*.

```
        OutputC= bestnetConstr(x0);
        c= OutputC(1:RunOptions.NumConstr,1);
        gradc= OutputC(RunOptions.NumConstr+1:...
            RunOptions.NumConstr*(length(x0)+1),1);

        ResultData(2*length(x0)+4:2*length(x0)+3+...
            RunOptions.NumConstr,Iteration)= c;
        ResultData(2*length(x0)+4+RunOptions.NumConstr:...
            (length(x0)+1)*(RunOptions.NumConstr+2)+1,...
                Iteration)= gradc;
```

Lo mismo ocurre con las restricciones y sus derivadas, cuyos valores son innecesarios para dar salida al módulo *ObjectiveFunction.m*, pero necesarios para el módulo *RestrictionFunction.m*.

```
ResultData((length(x0)+1)*(RunOptions.NumConstr+2)+2,...
    Iteration)= nan;
ResultData((length(x0)+1)*(RunOptions.NumConstr+2)+3,...
    Iteration)= nan;
ResultData((length(x0)+1)*(RunOptions.NumConstr+2)+4,...
    Iteration)= nan;
ResultData((length(x0)+1)*(RunOptions.NumConstr+2)+5,...
    Iteration)= nan;

tEndAna= toc(tStart);
tEndIter= tEndAna;

ResultData((length(x0)+1)*(RunOptions.NumConstr+2)+6,...
    Iteration)= tEndAna;
ResultData((length(x0)+1)*(RunOptions.NumConstr+2)+7,...
    Iteration)= tEndIter;
```

En los campos de *ResultData* relacionados con los análisis MEF el valor asignado es NaN (Not a Number).

El tiempo de iteración coincide con el tiempo de análisis.

```
else

    x0FEMUsefulData=FEMUsefulData(1:length(x0),:);

    minVal=min(x0FEMUsefulData(:));
    if minVal<0
        x0FEMUsefulData=x0FEMUsefulData+abs(minVal);
    end
```

Si no hay unas redes neuronales definidas, el programa procede al tratamiento de la información necesaria para construirlas.

Del conjunto de datos útiles para construir las redes neuronales, se extraen los valores de las variables de diseño y se busca el valor mínimo. Si dicho valor es negativo, su valor

absoluto es sumado a todos los demás valores. Este paso es necesario para el correcto tratamiento de la información.

```
normx0FEMUsefulData=zeros(1,size(x0FEMUsefulData,2));
for i=1:size(x0FEMUsefulData,2)
    normx0FEMUsefulData(1,i)=norm(x0FEMUsefulData...
        (1:length(x0),i));
end
```

Se halla la norma de todas las variables de diseño. Los datos se guardan en la variable *normx0FEMUsefulData*, que se inicializa como un vector fila de ceros para asignar el espacio de memoria necesario antes del bucle, acelerando la ejecución del mismo.

```
minPos=find(normx0FEMUsefulData==...
    min(normx0FEMUsefulData));
maxPos=find(normx0FEMUsefulData==...
    max(normx0FEMUsefulData));

minmaxFEMUsefulData=[FEMUsefulData(:,minPos),...
    FEMUsefulData(:,maxPos)];
```

Se buscan las posiciones de los valores máximos y mínimos de las normas y se crea la variable *minmaxFEMUsefulData* con las columnas del conjunto de datos útiles para construir las redes neuronales correspondientes a esas posiciones.

```
FEMUsefulData(:,minPos)=zeros(size(FEMUsefulData,1),...
    length(minPos));
FEMUsefulData(:,maxPos)=zeros(size(FEMUsefulData,1),...
    length(maxPos));
```

Los valores de las columnas del conjunto de datos útiles para construir las redes neuronales que han sido almacenadas en la variable *minmaxFEMUsefulData* se sustituyen por ceros en la variable *FEMUsefulData*.

```
FEMUsefulData= unique(FEMUsefulData','rows')';
FEMUsefulData= FEMUsefulData(:,2:size(FEMUsefulData,2));
FEMUsefulData= FEMUsefulData(:,randperm(size...
    (FEMUsefulData,2)));
ReadyFEMUsefulData=[minmaxFEMUsefulData,FEMUsefulData];
```

En estas últimas instrucciones de preparación de datos para la construcción de las redes neuronales, los datos de la variable *FEMUsefulData* son reordenados de la siguiente manera: las columnas de ceros se reducen a una, que se sitúa al principio. Se vuelve a definir *FEMUsefulData* con los datos de todas las columnas menos la primera, que posteriormente son permutadas aleatoriamente. Por último, se crea la variable *ReadyFEMUsefulData* mediante la unión de *minmaxFEMUsefulData* y *FEMUsefulData*. De esta forma, todo está preparado para la construcción de las redes neuronales.

```
Input=ReadyFEMUsefulData(1:length(x0),:);
TargetVol= ReadyFEMUsefulData(length(x0)+1:...
    2*length(x0)+1,:);
TargetC= ReadyFEMUsefulData(2*length(x0)+2:...
    size(ReadyFEMUsefulData,1),:);
```

Se definen los datos de entrada y los datos objetivo de las redes neuronales. Se creará una red neuronal para calcular el volumen y sus derivadas, y otra por cada conjunto restricción/derivadas de restricción (*TargetC1*, *TargetC2*,…).

El código actual está programado para tratar con una sola restricción (*TargetC*; tensión máxima de Von Mises), pero es fácilmente editable para que trabaje con más.

```
trainFcn= RunOptions.NNetTrainFcn;
hiddenLayerSizeVol= RunOptions.VolNNetnumNeurons;
hiddenLayerSizeConstr= RunOptions.ConstrNNetnumNeurons;
```

Se define la función que va a entrenar las redes neuronales, así como el número de neuronas de cada red, según lo que el usuario haya establecido en el módulo *ProblemData.m*.

```
netVol(1:RunOptions.NNetnumLoop)={zeros(1)};
performanceVol(1:RunOptions.NNetnumLoop)= 0;

for i=1:RunOptions.NNetnumLoop
    netVol{i}= feedforwardnet(hiddenLayerSizeVol,...
        trainFcn);
    netVol{i}= configure(netVol{i},Input,TargetVol);

    netVol{i}.input.processFcns= {'removeconstantrows',...
        'mapminmax'};
    netVol{i}.output.processFcns= {'removeconstantrows'...
        ,'mapminmax'};

    netVol{i}.divideFcn= 'divideblock';
    netVol{i}.divideMode= 'sample';

    netVol{i}.divideParam.trainRatio= 70/100;
    netVol{i}.divideParam.valRatio= 15/100;
    netVol{i}.divideParam.testRatio= 15/100;

    netVol{i}.performFcn= 'mse';

    netVol{i}= train(netVol{i},Input,TargetVol);
    nntraintool('close');

    Output= netVol{i}(Input);
    performanceVol(i)= perform(netVol{i},TargetVol,...
        Output);
end

bestVolmsePos= find(performanceVol==...
    min(performanceVol),1);
bestnetVol= netVol{bestVolmsePos};
```

El programa procede ahora a crear, en un bucle, un número de redes neuronales que permitan calcular el volumen y sus derivadas determinado por el usuario en el módulo *ProblemData.m*.

Cada red neuronal se crea con la función de entrenamiento y el número de neuronas definidos por el usuario en *ProblemData.m*, configura las dimensiones de su entrada y su salida para trabajar con los datos de entrada y los datos objetivo, establece las funciones que procesarán dichos datos para facilitar su manejo por la red neuronal, divide los datos mediante la función *divideblock* con propociones 70/15/15, entrena la red, y calcula su error de validación.

El programa elige la red neuronal con el menor error de validación para realizar los cálculos futuros.

```
OutputVol= bestnetVol(x0);
Volume= OutputVol(1,1);
GradVolume= OutputVol(2:length(x0)+1,1)';

ResultData(length(x0)+3,Iteration)= Volume;
ResultData(length(x0)+4:2*length(x0)+3,Iteration)=...
    GradVolume';
```

Se utiliza la red neuronal elegida para calcular el volumen y sus derivadas, que se almacenan en la variable *ResultData*.

```
netConstr(1:RunOptions.NNetnumLoop)={zeros(1)};
performanceConstr(1:RunOptions.NNetnumLoop)= 0;

for i=1:RunOptions.NNetnumLoop
    netConstr{i}= feedforwardnet(hiddenLayerSizeConstr,...
        trainFcn);
    netConstr{i}= configure(netConstr{i},Input,TargetC);

    netConstr{i}.input.processFcns=...
        {'removeconstantrows','mapminmax'};
    netConstr{i}.output.processFcns=...
        {'removeconstantrows','mapminmax'};

    netConstr{i}.divideFcn= 'divideblock';
    netConstr{i}.divideMode= 'sample';

    netConstr{i}.divideParam.trainRatio= 70/100;
    netConstr{i}.divideParam.valRatio= 15/100;
    netConstr{i}.divideParam.testRatio= 15/100;

    netConstr{i}.performFcn= 'mse';

    netConstr{i}= train(netConstr{i},Input,TargetC);
    nntraintool('close');

    Output= netConstr{i}(Input);
    performanceConstr(i)= perform(netConstr{i},TargetC,...
        Output);
end

bestConstrmsePos= find(performanceConstr==...
```

```
            min(performanceConstr),1);
        bestnetConstr= netConstr{bestConstrmsePos};

        OutputC= bestnetConstr(x0);
        c= OutputC(1:RunOptions.NumConstr,1);
        gradc= OutputC(RunOptions.NumConstr+1:...
            RunOptions.NumConstr*(length(x0)+1),1);

        ResultData(2*length(x0)+4:2*length(x0)+3+...
            RunOptions.NumConstr,Iteration)= c;
        ResultData(2*length(x0)+4+RunOptions.NumConstr:...
            (length(x0)+1)*(RunOptions.NumConstr+2)+1,...
                Iteration)= gradc;
```

Se sigue exactamente el mismo proceso para las restricciones. En el caso de la existencia de más de una restricción, se deberá repetir este trozo de código tantas veces como sea necesario, dando lugar a varias redes neuronales (*bestnetConstr1*, *bestnetConstr2*,…).

```
        ResultData((length(x0)+1)*(RunOptions.NumConstr+2)+2,...
            Iteration)= nan;
        ResultData((length(x0)+1)*(RunOptions.NumConstr+2)+3,...
            Iteration)= nan;
        ResultData((length(x0)+1)*(RunOptions.NumConstr+2)+4,...
            Iteration)= nan;
        ResultData((length(x0)+1)*(RunOptions.NumConstr+2)+5,...
            Iteration)= nan;
        tEndAna= toc(tStart);
        tEndIter= tEndAna;

        ResultData((length(x0)+1)*(RunOptions.NumConstr+2)+6,...
            Iteration)= tEndAna;
        ResultData((length(x0)+1)*(RunOptions.NumConstr+2)+7,...
            Iteration)= tEndIter;

        thereIsNNet= 1;

    end
```

En los campos de *ResultData* relacionados con los análisis MEF se almacenan valores NaN (Not a Number). El tiempo de iteración coincide con el tiempo de análisis. Por último, se hace saber al programa que ya tiene a su disposición las redes neuronales necesarias para hacer los cálculos pertinentes.

Esta sección del módulo *ObjectiveFunction.m* se corresponde al "bloque de redes neuronales".

```
else

    tStart= tic;

    if numNNetUses>=RunOptions.maxNNetUses
        numFEMUses=numFEMUses+1;
    end

    IterOrigin= RunOptions.ObjFunFEM;
    DataOrigin= IterOrigin;

    ResultData(1,Iteration)= IterOrigin;
    ResultData(2,Iteration)= DataOrigin;
    ResultData(3:length(x0)+2,Iteration)= x0';
```

Si no hay repetición de datos ni es posible utilizar las redes neuronales para hacer cálculos, se procede a la realización del análisis MEF.

Si se ha alcanzado el número máximo de usos de las redes neuronales definido por el usuario en *ProblemData.m*, el contador de análisis MEF se incrementa en 1.

El origen de los datos coincide con el origen de la iteración.

```
    try

        Param.ModeBatch=1;

        Param.Contour=1;

        SensAnaIter=[];
        SensAnaGlob=[];
        SensAnaGlob.Param.NDesVar=0;
        SensAnaGlob.Param.DesVarFile=0;
        SensAnaGlob.Param.AnaliticVF=0;
        SensAnaGlob.Param.VFEmptyArea=100;
        SensAnaGlob.Param.LoadCaseVF=2;
        SensAnaGlob.Param.LoadCaseSens=1;
        SensAnaGlob.Param.InterpType=[0 0 1];
        SensAnaGlob.Param.DomainType=zeros(1,3);
        Reference.SensAnaGlob.Param.DesVarIncGeos=zeros(1,4);
```

```
IniConstants;

Reference.Parameters=x0;
ProblemData('ProcessGeo');
run(SensAnaGlob.Param.DesVarFile);
ReCalcAll;

ProblemData('MeshParam');
if BatchInfo.SquareMesh
    dummymin=min(BatchInfo.MeshCoords(:,1));
    dummymax=max(BatchInfo.MeshCoords(:,2));
    BatchInfo.MeshCoords=[dummymin dummymax;...
        dummymin dummymax];
end

Source=2;
MainGE(Source)
```

El análisis MEF es la sección del programa donde se pueden producir errores con más facilidad (por ejemplo, si existen solapes en las líneas que forman el contorno de la geometría propuesta por el programa de optimización). Por tanto, toda la ejecución del análisis MEF está dentro de una instrucción *try…catch…end*.

El programa debe ser ejecutado sin el uso de la interfaz de usuario, y debe realizar cálculos en los elementos pertenecientes al contorno de la geometría.

Se inicializan las variables relacionadas con los análisis MEF y se dibuja la nueva geometría según los valores de las variables de diseño sugeridas por el proceso de optimización.

Se definen los parámetros del proceso de mallado, según el contenido de *`ProblemData.m`*, y se ejecuta el calculador *`MainGE`* desde el programa de optimización (*`Source=2`*).

```
sa_deltaV(iMesh,0);

ResultData(length(x0)+3,Iteration)=...
    Iter(iMesh).Volume.Vtotal;
ResultData(length(x0)+4:2*length(x0)+3,Iteration)=...
    Reference.dV';

Volume= Iter(iMesh).Volume.Vtotal;
GradVolume= Reference.dV;
```

Se calculan los valores del volumen y sus derivadas y se almacenan en la variable *ResultData*.

```
sa_StressVM(iMesh,0);

c=[Reference.MaxStressVM*Geo.Material.StressFOS-...
    Geo.Material.MaxStress, ];
gradc=[Reference.dStressOfMaxStressVM, ];

if RunOptions.NumConstr ~= length(c)

    fprintf('&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&...
        &&&&&&\r\n')
    fprintf('CHECK THE NUMBER OF CONSTRAINTS AND TRY...
        AGAIN\r\n')
    fprintf('&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&...
        &&&&&&\r\n')

    error('Number of constraints')

end

RunOptions.NumConstr= length(c);

ResultData(2*length(x0)+4:2*length(x0)+3+...
    RunOptions.NumConstr,Iteration)= c';
ResultData(2*length(x0)+4+RunOptions.NumConstr:...
    (length(x0)+1)*(RunOptions.NumConstr+2)+1,...
        Iteration)= gradc';
```

Se calculan los valores de las restricciones y sus derivadas y se almacenan en la variable *ResultData*.

Si se desea añadir más restricciones al problema, las variables *c* y *gradc* están preparadas. Es importante saber que el número de restricciones existentes debe ser definido en el módulo *ProblemData.m* por el usuario. Si ese número no coincide con la longitud de la variable *c*, como medida de seguridad, el programa se detiene e informa al usuario de lo sucedido.

```
ResultData((length(x0)+1)*(RunOptions.NumConstr+2)+4,...
    Iteration)= CalcMesh(iMesh).NNodes*2;
ResultData((length(x0)+1)*(RunOptions.NumConstr+2)+5,...
    Iteration)= CalcMesh(iMesh).NumEle;
```

Se almacenan en la variable *ResultData* el número de elementos y el número de grados de libertad.

Los valores de las otras dos variables relacionadas con los análisis MEF (número de malla y error en norma energética en dicha malla) se obtienen de la rutina *ErrorCalc.m*. A continuación se expone el fragmento que interesa de dicha rutina:

```
global Elm Iter Nod Param Prob MatK CalcMesh Geo Iteration...
    IniValRes RunOptions BatchInfo ResultData...
    GlobOptIsDone SolutionIt
```

Éstas son las variables globales de la rutina, necesarias para su correcto funcionamiento.

```
    if iLoadCase ~= length(BatchInfo.LoadCase2Solve) &&...
        (RelatError_Control(iLoadCase)<=...
            BatchInfo.TargErr(1,length(IniValRes.x0)+1)||...
                iMesh==BatchInfo.MxNIter)

        if GlobOptIsDone==0
            ResultData((length(IniValRes.x0)+1)*...
                (RunOptions.NumConstr+2)+2,Iteration)= iMesh;
            ResultData((length(IniValRes.x0)+1)*...
                (RunOptions.NumConstr+2)+3,Iteration)=...
                    RelatError_Control(iLoadCase);
        else
            ResultData((length(IniValRes.x0)+1)*...
                (RunOptions.NumConstr+2)+2,SolutionIt)= iMesh;
            ResultData((length(IniValRes.x0)+1)*...
                (RunOptions.NumConstr+2)+3,SolutionIt)=...
                    RelatError_Control(iLoadCase);
        end

    end
```

Si el error en norma energética en la malla es igual o menor al máximo definido por el usuario en el módulo *ProblemData.m*, o bien el número de iteraciones de malla del proceso h-adaptativo ha alcanzado el máximo definido por el usuario, se almacena en el campo correspondiente de la variable *ResultData* los valores de ambos, en la columna de la iteración correspondiente (si el proceso de optimización no ha terminado, es decir,

si *GlobOptIsDone*=0) o en la columna de la iteración solución (si el proceso de optimización ha finalizado, es decir, si *GlobOptIsDone*=1).

Volviendo al módulo *ObjectiveFunction.m*:

```
tEndAna= toc(tStart);
tEndIter= tEndAna;

ResultData((length(x0)+1)*(RunOptions.NumConstr+2)+6,...
    Iteration)= tEndAna;
ResultData((length(x0)+1)*(RunOptions.NumConstr+2)+7,...
    Iteration)= tEndIter;
```

El tiempo de iteración coincide con el tiempo de análisis.

```
catch ME

    if ~exist(fullfile(ResultsPath,'Error Info'),'dir')
        mkdir(fullfile(ResultsPath,'Error Info'));
    end

    if ~exist(fullfile(ResultsPath,'Error Info','Detailed...
        Error Geometries'),'dir')
        mkdir(fullfile(ResultsPath,'Error Info','Detailed...
            Error Geometries'));
    end

    if ~exist(fullfile(ResultsPath,'Error Info','Scaled...
        Error Geometries'),'dir')
        mkdir(fullfile(ResultsPath,'Error Info','Scaled...
            Error Geometries'));
    end
```

Si hay algún error durante el análisis MEF, el programa salta al bloque *catch* y crea los directorios necesarios para almacenar toda la información de dicho error.

```
close all;

DMesh(-1,1,0,0);
DGeo(0,0,0);

saveas(gcf,fullfile(ResultsPath,'Error Info','Detailed...
    Error Geometries',strcat(['Geometry' ' '...
        num2str(Iteration) ' Detailed' '.jpg'])));

axis([RunOptions.ScaledAxisXmin...
    RunOptions.ScaledAxisXmax...
        RunOptions.ScaledAxisYmin...
            RunOptions.ScaledAxisYmax]);
saveas(gcf,fullfile(ResultsPath,'Error Info','Scaled...
    Error Geometries',strcat(['Geometry' ' '...
        num2str(Iteration) ' Scaled' '.jpg'])));
```

El programa dibuja en una figura la malla y la geometría que han dado lugar al error y la almacena en la carpeta correspondiente. Se generan dos tipos de figuras: una en la que la geometría y la malla ocupan siempre todo el espacio, y otra en la que se dibujan según unos ejes fijos definidos por el usuario en *ProblemData.m*, lo que permite comparar visiblemente las distintas geometrías.

```
Error_output = fullfile(ResultsPath,'Error Info','Error...
    Data.txt');
fid = fopen(Error_output,'a');
fprintf(fid,'&&&&&&&&&& ITERATION %5G\r\n',Iteration);
fprintf(fid,'%s\r\n',ME.identifier);
fprintf(fid,'%s\r\n\r\n',ME.message);
fprintf(fid,'FAILURES:\r\n');
    for e=1:length(ME.stack)
        fprintf(fid,'%s at %i\r\n',...
            ME.stack(e).name,ME.stack(e).line);
    end
fprintf(fid,'\r\n');
fprintf(fid,'X value=\r\n');
fprintf(fid,'%25G\r\n', x0');
fprintf(fid,'\r\n\r\n');
fclose(fid);
```

Se crea el archivo *Error Data.txt*, donde se escribe el número de iteración donde se produce el error, el identificador de éste, el mensaje correspondiente, la lista de fallos (en qué línea de qué funciones se producen los fallos), y los valores de las variables de diseño con los que ocurre.

```
        ResultData(length(x0)+3,Iteration)= nan;
        ResultData(length(x0)+4:2*length(x0)+3,Iteration)=...
            nan(1,length(x0))';

        Volume= nan;
        GradVolume= nan(1,length(x0));
```

Al producirse un error, los valores del volumen y sus derivadas son desconocidos, luego se les asignan valores NaN (Not a Number), con los que el programa de optimización puede seguir trabajando.

```
        ResultData(2*length(x0)+4:2*length(x0)+3+...
            RunOptions.NumConstr,Iteration)=...
                [Geo.Material.MaxStress, ]';
        ResultData(2*length(x0)+4+RunOptions.NumConstr:...
            (length(x0)+1)*(RunOptions.NumConstr+2)+1,...
                Iteration)= nan(1,RunOptions.NumConstr*...
                    length(x0))';
```

Lo mismo ocurre con los valores de las restricciones, pero a éstas se les asigna “artificialmente” un valor que haga que, tanto el proceso de optimización como los cálculos con redes neuronales, no las consideren como posible solución del problema. En el caso de la restricción tensión máxima de Von Mises, el valor asignado es la tensión máxima admisible. Si se desea definir más restricciones, se deberán añadir los valores correspondientes.

```
        ResultData((length(x0)+1)*(RunOptions.NumConstr+2)+2,...
            Iteration)= nan;
        ResultData((length(x0)+1)*(RunOptions.NumConstr+2)+3,...
            Iteration)= nan;

        ResultData((length(x0)+1)*(RunOptions.NumConstr+2)+4,...
            Iteration)= nan;
        ResultData((length(x0)+1)*(RunOptions.NumConstr+2)+5,...
            Iteration)= nan;
        tEndAna= toc(tStart);
        tEndIter= tEndAna;
        ResultData((length(x0)+1)*(RunOptions.NumConstr+2)+6,...
            Iteration)= tEndAna;
        ResultData((length(x0)+1)*(RunOptions.NumConstr+2)+7,...
            Iteration)= tEndIter;
    end
end
```

En los campos restantes relacionados con el análisis MEF de la variable *ResultData* se escriben valores NaN (Not a Number).

El tiempo de análisis coincide con el tiempo de iteración.

Esta sección del módulo *ObjectiveFunction.m* es el "bloque de análisis MEF".

```
if (ResultData(2*length(x0)+4:2*length(x0)+3+...
    RunOptions.NumConstr,Iteration)<=10*eps*...
        ones(RunOptions.NumConstr,1)) && (ResultData(1,...
            Iteration)==RunOptions.ObjFunFEM)

    close all;
    DMesh(-1,1,0,0);
    DGeo(0,0,0);

    saveas(gcf,fullfile(ResultsPath,'Detailed Acceptable...
        Geometries',strcat(['Geometry' ' ' num2str(Iteration)...
            ' Detailed' '.jpg'])));

    axis([RunOptions.ScaledAxisXmin...
        RunOptions.ScaledAxisXmax...
            RunOptions.ScaledAxisYmin...
                RunOptions.ScaledAxisYmax]);
    saveas(gcf,fullfile(ResultsPath, 'Scaled Acceptable...
        Geometries',strcat(['Geometry' ' ' num2str(Iteration)...
            ' Scaled' '.jpg'])));

end
```

Si la geometría de una determinada iteración es aceptable (cumple todas las restricciones) y sus datos provienen de un análisis MEF, dibuja la geometría y la malla, y crea las figuras explicadas anteriormente.

```
Data_output = fullfile(ResultsPath,'Result Data.txt');
fid = fopen(Data_output,'a');
fprintf(fid,'&&&&&&&&&& ITERATION=\r\n');
fprintf(fid,'%25G\r\n\r\n', Iteration);
fprintf(fid,'Iteration Origin=\r\n');
fprintf(fid,'%25G\r\n\r\n', ResultData(1,Iteration));
fprintf(fid,'X=\r\n');
fprintf(fid,'%25G\r\n', ResultData(3:length(x0)+2,Iteration));
fprintf(fid,'\r\n');
fprintf(fid,'Volume=\r\n');
```

```
fprintf(fid,'%25G\r\n\r\n', ResultData(length(x0)+3,Iteration));
fprintf(fid,'Constraints=\r\n');
fprintf(fid,'%25G\r\n', ResultData(2*length(x0)+4:2*length(x0)+...
    3+RunOptions.NumConstr,Iteration));
fprintf(fid,'\r\n');
fprintf(fid,'Mesh Error (%%)=\r\n');
fprintf(fid,'%25G\r\n\r\n', ResultData((length(x0)+1)*...
    (RunOptions.NumConstr+2)+3,Iteration));
fprintf(fid,'Iteration Time (sec)=\r\n');
fprintf(fid,'%25G\r\n\r\n\r\n', ResultData((length(x0)+1)*...
    (RunOptions.NumConstr+2)+7,Iteration));
fclose(fid);
```

El programa crea el archivo *Result Data.txt*, donde se almacena información útil de cada iteración: el número de iteración, el origen de la iteración, los valores de las variables de diseño, los valores del volumen y las restricciones, el error en norma energética en la malla y el tiempo de iteración.

```
if numNNetUses>=RunOptions.maxNNetUses
    UseNNet=0;
end

if numFEMUses>=RunOptions.maxFEMUses
    thereIsNNet= 0;
    numNNetUses= 0;
    numFEMUses= 0;
end

save(fullfile(ProblemPath,'Stored ResultData',...
    strcat(['ResultData' ' ' num2str(ActualDate) '.mat'])),...
        'ResultData');
```

Si el número de usos de unas determinadas redes neuronales alcanza el máximo definido por el usuario en el módulo *`ProblemData.m`*, el programa deja de utilizarlas.

Si, una vez que el programa ha entrado en la dinámica de las redes neuronales, el número de usos del análisis MEF alcanza el máximo definido por el usuario en el módulo *`ProblemData.m`*, el programa se prepara para volver a utilizar redes neuronales.

Por último, al final de cada iteración, el programa guarda la variable *`ResultData`* en la carpeta *Stored ResultData*, con la fecha y hora del comienzo del análisis en curso.

Esta última sección del módulo *ObjectiveFunction.m* es el "bloque de presentación de resultados".

B.2.3. *RestrictionFunction.m*

Es uno de los dos módulos que componen el proceso de optimización. Está presente en el directorio principal del programa.

Se encarga de calcular el valor de las restricciones y sus derivadas para unos valores determinados de las variables de diseño.

El código es casi idéntico al del módulo *ObjectiveFunction.m*, de modo que sólo se explicarán las diferencias:

```
function [c,ceq,gradc,gradceq]= RestrictionFunction(x0)

global Geo Param FEMGui Nod Elm Errores BatchInfo Iter iMesh...
    CalcMesh GlobalMatKData GlobalIntersData SensAnaGlob...
    SensAnaIter MatK Reference ResultsPath RunOptions Iteration...
    ResultData PrevResData ProblemPath ActualDate...
    thereIsNNet UseNNet numNNetUses numFEMUses...
    normx0FEMUsefulData bestnetVol bestnetConstr

Constants;
Iteration= Iteration+1;
```

Se define la función *RestrictionFunction.m*, cuyas variables de salida son: restricciones de desigualdad (c), restricciones de igualdad (ceq, inexistentes), derivadas de restricciones de desigualdad (gradc), y derivadas de restricciones de igualdad (gradceq, inexistentes).

```
Repeat= 0;

if RunOptions.UsePrevData==1
    RepeatDataPool= [PrevResData,ResultData(:,1:Iteration-1)];
else
    RepeatDataPool= ResultData(:,1:Iteration-1);
end
```

Sólo es necesario considerar dos situaciones a la hora de definir la variable *RepeatDataPool*.

```
Check= RepeatDataPool(3:length(x0)+2,:)-x0'*...
    ones(1,size(RepeatDataPool,2));
for i=1:size(RepeatDataPool,2)
    if norm(Check(1:length(x0),i))<=RunOptions.TolX*norm(x0)
        Repeat= 1;
        RepeatIt= i;
        break
    end
end


if RunOptions.UseNNet==1 && Repeat==0

    NNetDataPool= CreateNNetDataPool(RunOptions.DataPoolOrigin);

    originVector= NNetDataPool(2,:);
    FEMDataPos= find(originVector==RunOptions.ObjFunFEM|...
        originVector==RunOptions.ResFunFEM);
    FEMDataMatrix= NNetDataPool(3:(length(x0)+1)*...
        (RunOptions.NumConstr+2)+1,FEMDataPos);
    FEMUsefulData= unique(FEMDataMatrix','rows')';
    if thereIsNNet==0 && size(FEMUsefulData,2)>=...
        RunOptions.NNetminInputs
        UseNNet= 1;
    elseif thereIsNNet==1 && (norm(x0)<...
        min(normx0FEMUsefulData)||norm(x0)>...
            max(normx0FEMUsefulData))
        UseNNet= 0;
        thereIsNNet= 0;
        numNNetUses= 0;
    end

end

if Repeat==1

    tStartIter= tic;

    IterOrigin= RunOptions.ResFunRepeat;
    ResultData(1,Iteration)= IterOrigin;

    ResultData(2:(length(x0)+1)*(RunOptions.NumConstr+2)+6,...
        Iteration)=RepeatDataPool(2:(length(x0)+1)*...
            (RunOptions.NumConstr+2)+6, RepeatIt);

    c=ResultData(2*length(x0)+4:2*length(x0)+3+...
```

```
        RunOptions.NumConstr,Iteration)';
    ceq=[ ];

    gradcprov= ResultData(2*length(x0)+4+RunOptions.NumConstr:...
        (length(x0)+1)*(RunOptions.NumConstr+2)+1,...
            Iteration)';
    gradc=zeros(length(x0),RunOptions.NumConstr);
    for cons=1:RunOptions.NumConstr
        gradc(1:length(x0),cons)=gradcprov(1,1+(cons-1)*...
            length(x0):cons*length(x0))';
    end
    gradceq=[ ];
```

Aquí la diferencia con el módulo *ObjectiveFunction.m* es notable. Las variables de salida son *c*, *ceq*, *gradc*, *gradceq*. Al igual que antes, la repetición de resultados hace que dichos valores se extraigan de la variable *ResultData*, pero los correspondientes a las derivadas de las restricciones de desigualdad han de seguir un proceso de redimensionamiento, para que sean aceptadas como salidas por el programa de optimización: *gradc* ha de tener las dimensiones [número de variables de diseño x número de restricciones de desigualdad].

```
    tEndIter= toc(tStartIter);
    ResultData((length(x0)+1)*(RunOptions.NumConstr+2)+7,...
        Iteration)= tEndIter;

elseif UseNNet==1

    tStart= tic;

    numNNetUses=numNNetUses+1;

    IterOrigin= RunOptions.ResFunNNet;
    DataOrigin= IterOrigin;

    ResultData(1,Iteration)= IterOrigin;
    ResultData(2,Iteration)= DataOrigin;
    ResultData(3:length(x0)+2,Iteration)= x0';

    if thereIsNNet==1

        OutputVol= bestnetVol(x0);
        Volume= OutputVol(1,1);
        GradVolume= OutputVol(2:length(x0)+1,1)';

        ResultData(length(x0)+3,Iteration)= Volume;
```

```
        ResultData(length(x0)+4:2*length(x0)+3,Iteration)=...
            GradVolume';


        OutputC= bestnetConstr(x0);
        c= OutputC(1:RunOptions.NumConstr,1)';
        ceq=[ ];

        gradcprov= OutputC(RunOptions.NumConstr+1:...
            RunOptions.NumConstr*(length(x0)+1),1)';
        gradc=zeros(length(x0),RunOptions.NumConstr);
        for cons=1:RunOptions.NumConstr
            gradc(1:length(x0),cons)=gradcprov(1,1+(cons-1)*...
                length(x0):cons*length(x0))';
        end
        gradceq=[ ];

        ResultData(2*length(x0)+4:2*length(x0)+3+...
            RunOptions.NumConstr,Iteration)= c';
        ResultData(2*length(x0)+4+RunOptions.NumConstr:...
            (length(x0)+1)*(RunOptions.NumConstr+2)+1,...
                Iteration)= gradcprov';
```

Las redes neuronales deben dar salida a *ceq*, *gradceq* (aunque sea un vector vacío) y la información de *gradc* ha de ser procesada de la misma forma que antes.

```
        ResultData((length(x0)+1)*(RunOptions.NumConstr+2)+2,...
            Iteration)= nan;
        ResultData((length(x0)+1)*(RunOptions.NumConstr+2)+3,...
            Iteration)= nan;
        ResultData((length(x0)+1)*(RunOptions.NumConstr+2)+4,...
            Iteration)= nan;
        ResultData((length(x0)+1)*(RunOptions.NumConstr+2)+5,...
            Iteration)= nan;

        tEndAna= toc(tStart);
        tEndIter= tEndAna;

        ResultData((length(x0)+1)*(RunOptions.NumConstr+2)+6,...
            Iteration)= tEndAna;
        ResultData((length(x0)+1)*(RunOptions.NumConstr+2)+7,...
            Iteration)= tEndIter;

    else

        x0FEMUsefulData=FEMUsefulData(1:length(x0),:);
```

```
minVal=min(x0FEMUsefulData(:));
if minVal<0
    x0FEMUsefulData=x0FEMUsefulData+abs(minVal);
end

normx0FEMUsefulData=zeros(1,size(x0FEMUsefulData,2));
for i=1:size(x0FEMUsefulData,2)
    normx0FEMUsefulData(1,i)=norm(x0FEMUsefulData(1:...
        length(x0),i));
end

minPos=find(normx0FEMUsefulData==...
    min(normx0FEMUsefulData));
maxPos=find(normx0FEMUsefulData==...
    max(normx0FEMUsefulData));

minmaxFEMUsefulData=[FEMUsefulData(:,minPos),...
    FEMUsefulData(:,maxPos)];

FEMUsefulData(:,minPos)=zeros(size(FEMUsefulData,1),...
    length(minPos));
FEMUsefulData(:,maxPos)=zeros(size(FEMUsefulData,1),...
    length(maxPos));

FEMUsefulData= unique(FEMUsefulData','rows')';
FEMUsefulData= FEMUsefulData(:,2:size(FEMUsefulData,2));
FEMUsefulData= FEMUsefulData(:,randperm(size(...
    FEMUsefulData,2)));
ReadyFEMUsefulData=[minmaxFEMUsefulData,FEMUsefulData];

Input=ReadyFEMUsefulData(1:length(x0),:);
TargetVol= ReadyFEMUsefulData(length(x0)+1:...
    2*length(x0)+1,:);
TargetC= ReadyFEMUsefulData(2*length(x0)+2:...
    size(ReadyFEMUsefulData,1),:);

trainFcn= RunOptions.NNetTrainFcn;
hiddenLayerSizeVol= RunOptions.VolNNetnumNeurons;
hiddenLayerSizeConstr= RunOptions.ConstrNNetnumNeurons;
netVol(1:RunOptions.NNetnumLoop)={zeros(1)};
performanceVol(1:RunOptions.NNetnumLoop)= 0;

for i=1:RunOptions.NNetnumLoop
    netVol{i}= feedforwardnet(hiddenLayerSizeVol,...
        trainFcn);
    netVol{i}= configure(netVol{i},Input,TargetVol);

    netVol{i}.input.processFcns= {'removeconstantrows',...
        'mapminmax'};
    netVol{i}.output.processFcns={'removeconstantrows',...
        'mapminmax'};
```

```
        netVol{i}.divideFcn= 'divideblock';
        netVol{i}.divideMode= 'sample';

        netVol{i}.divideParam.trainRatio= 70/100;
        netVol{i}.divideParam.valRatio= 15/100;
        netVol{i}.divideParam.testRatio= 15/100;

        netVol{i}.performFcn= 'mse';

        netVol{i}= train(netVol{i},Input,TargetVol);
        nntraintool('close');

        Output= netVol{i}(Input);
        performanceVol(i)= perform(netVol{i},TargetVol,...
            Output);
    end

    bestVolmsePos= find(performanceVol==...
        min(performanceVol),1);
    bestnetVol= netVol{bestVolmsePos};

    OutputVol= bestnetVol(x0);
    Volume= OutputVol(1,1);
    GradVolume= OutputVol(2:length(x0)+1,1)';

    ResultData(length(x0)+3,Iteration)= Volume;
    ResultData(length(x0)+4:2*length(x0)+3,...
        Iteration)= GradVolume';

    netConstr(1:RunOptions.NNetnumLoop)={zeros(1)};
    performanceConstr(1:RunOptions.NNetnumLoop)= 0;

    for i=1:RunOptions.NNetnumLoop
        netConstr{i}= feedforwardnet(hiddenLayerSizeConstr,...
            trainFcn);
        netConstr{i}= configure(netConstr{i},Input,TargetC);

        netConstr{i}.input.processFcns=...
            {'removeconstantrows','mapminmax'};
        netConstr{i}.output.processFcns=...
            {'removeconstantrows','mapminmax'};

        netConstr{i}.divideFcn= 'divideblock';
        netConstr{i}.divideMode= 'sample';

        netConstr{i}.divideParam.trainRatio= 70/100;
        netConstr{i}.divideParam.valRatio= 15/100;
        netConstr{i}.divideParam.testRatio= 15/100;

        netConstr{i}.performFcn= 'mse';

        netConstr{i}= train(netConstr{i},Input,TargetC);
```

```
        nntraintool('close');

        Output= netConstr{i}(Input);
        performanceConstr(i)= perform(netConstr{i},TargetC,...
            Output);
    end

    bestConstrmsePos= find(performanceConstr==...
        min(performanceConstr),1);
    bestnetConstr= netConstr{bestConstrmsePos};

    OutputC= bestnetConstr(x0);
    c= OutputC(1:RunOptions.NumConstr,1);
    ceq=[ ];

    gradcprov= OutputC(RunOptions.NumConstr+1:...
        RunOptions.NumConstr*(length(x0)+1),1)';
    gradc=zeros(length(x0),RunOptions.NumConstr);
    for cons=1:RunOptions.NumConstr
        gradc(1:length(x0),cons)=gradcprov(1,1+(cons-1)*...
            length(x0):cons*length(x0))';
    end
    gradceq=[ ];

    ResultData(2*length(x0)+4:2*length(x0)+3+...
        RunOptions.NumConstr,Iteration)= c';
    ResultData(2*length(x0)+4+RunOptions.NumConstr:...
        (length(x0)+1)*(RunOptions.NumConstr+2)+1,...
            Iteration)= gradcprov';
```

Las redes neuronales deben dar salida a *ceq*, *gradceq* (aunque sea un vector vacío) y la información de *gradc* ha de ser procesada de la misma forma que antes.

```
    ResultData((length(x0)+1)*(RunOptions.NumConstr+2)+2,...
        Iteration)= nan;
    ResultData((length(x0)+1)*(RunOptions.NumConstr+2)+3,...
        Iteration)= nan;
    ResultData((length(x0)+1)*(RunOptions.NumConstr+2)+4,...
        Iteration)= nan;
    ResultData((length(x0)+1)*(RunOptions.NumConstr+2)+5,...
        Iteration)= nan;

    tEndAna= toc(tStart);
    tEndIter= tEndAna;

    ResultData((length(x0)+1)*(RunOptions.NumConstr+2)+6,...
        Iteration)= tEndAna;
    ResultData((length(x0)+1)*(RunOptions.NumConstr+2)+7,...
```

```
            Iteration)= tEndIter;

        thereIsNNet= 1;

    end

else

    tStart= tic;

    if numNNetUses>=RunOptions.maxNNetUses
        numFEMUses=numFEMUses+1;
    end

    IterOrigin= RunOptions.ResFunFEM;
    DataOrigin= IterOrigin;

    ResultData(1,Iteration)= IterOrigin;
    ResultData(2,Iteration)= DataOrigin;
    ResultData(3:length(x0)+2,Iteration)= x0';

try

        Param.ModeBatch=1;

        Param.Contour=1;

        SensAnaIter=[];
        SensAnaGlob=[];
        SensAnaGlob.Param.NDesVar=0;
        SensAnaGlob.Param.DesVarFile=0;
        SensAnaGlob.Param.AnaliticVF=0;
        SensAnaGlob.Param.VFEmptyArea=100;
        SensAnaGlob.Param.LoadCaseVF=2;
        SensAnaGlob.Param.LoadCaseSens=1;
        SensAnaGlob.Param.InterpType=[0 0 1];
        SensAnaGlob.Param.DomainType=zeros(1,3);
        Reference.SensAnaGlob.Param.DesVarIncGeos=zeros(1,4);

        IniConstants;

        Reference.Parameters=x0;
        ProblemData('ProcessGeo');
        run(SensAnaGlob.Param.DesVarFile);
        ReCalcAll;

        ProblemData('MeshParam');
        if BatchInfo.SquareMesh
            dummymin=min(BatchInfo.MeshCoords(:,1));
            dummymax=max(BatchInfo.MeshCoords(:,2));
            BatchInfo.MeshCoords=[dummymin dummymax;...
                dummymin dummymax];
```

```
end

Source=2;
MainGE(Source)

sa_deltaV(iMesh,0);

ResultData(length(x0)+3,Iteration)=...
    Iter(iMesh).Volume.Vtotal;
ResultData(length(x0)+4:2*length(x0)+3,...
    Iteration)= Reference.dV';

Volume= Iter(iMesh).Volume.Vtotal;
GradVolume= Reference.dV;

sa_StressVM(iMesh,0);

c=[Reference.MaxStressVM*Geo.Material.StressFOS-...
    Geo.Material.MaxStress, ];
ceq=[ ];
if RunOptions.NumConstr ~= length(c)

    fprintf('&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&...
        &&&&&&\r\n')
    fprintf('CHECK THE NUMBER OF CONSTRAINTS AND TRY...
        AGAIN\r\n')
    fprintf('&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&...
        &&&&&&\r\n')

    error('Number of constraints')

end
```

El programa comprueba que el número de restricciones presentes en el código del módulo *ObjectiveFunction.m* concuerda con el correspondiente a éste módulo. Es una forma de asegurarse de que el número de restricciones definidas es igual en ambos módulos, detectando así posibles errores en la edición del programa.

```
gradcprov= [Reference.dStressOfMaxStressVM, ];
gradc=zeros(length(x0),RunOptions.NumConstr);
for cons=1:RunOptions.NumConstr

    gradc(1:length(x0),cons)=gradcprov(1,1+...
        (cons-1)*length(x0):cons*length(x0))';

end
gradceq=[ ];
```

```
        ResultData(2*length(x0)+4:2*length(x0)+3+...
            RunOptions.NumConstr,Iteration)= c';
        ResultData(2*length(x0)+4+RunOptions.NumConstr:...
            (length(x0)+1)*(RunOptions.NumConstr+2)+1,...
                Iteration)= gradcprov';
```

El análisis MEF debe dar salida a *ceq*, *gradceq* (aunque sea un vector vacío) y la información de *gradc* ha de ser procesada de la misma forma que antes.

```
        ResultData((length(x0)+1)*(RunOptions.NumConstr+2)+4,...
            Iteration)=CalcMesh(iMesh).NNodes*2;
        ResultData((length(x0)+1)*(RunOptions.NumConstr+2)+5,...
            Iteration)=CalcMesh(iMesh).NumEle;

        tEndAna= toc(tStart);
        tEndIter= tEndAna;

        ResultData((length(x0)+1)*(RunOptions.NumConstr+2)+6,...
            Iteration)= tEndAna;
        ResultData((length(x0)+1)*(RunOptions.NumConstr+2)+7,...
            Iteration)= tEndIter;


catch ME

        if ~exist(fullfile(ResultsPath,'Error Info'),'dir')
            mkdir(fullfile(ResultsPath,'Error Info'));
        end

        if ~exist(fullfile(ResultsPath,'Error Info','Detailed...
            Error Geometries'),'dir')
            mkdir(fullfile(ResultsPath,'Error Info','Detailed...
                Error Geometries'));
        end

        if ~exist(fullfile(ResultsPath,'Error Info','Scaled...
            Error Geometries'),'dir')
            mkdir(fullfile(ResultsPath,'Error Info','Scaled...
                Error Geometries'));
        end

        close all;

        DMesh(-1,1,0,0);
        DGeo(0,0,0);

        saveas(gcf,fullfile(ResultsPath,'Error Info','Detailed...
```

```
        Error Geometries',strcat(['Geometry' ' '...
            num2str(Iteration) ' Detailed' '.jpg'])));

    axis([RunOptions.ScaledAxisXmin...
        RunOptions.ScaledAxisXmax...
            RunOptions.ScaledAxisYmin...
                RunOptions.ScaledAxisYmax]);
    saveas(gcf,fullfile(ResultsPath,'Error Info','Scaled...
        Error Geometries',strcat(['Geometry' ' '...
            num2str(Iteration) ' Scaled' '.jpg'])));

    Error_output = fullfile(ResultsPath,'Error Info',...
        'Error Data.txt');
    fid = fopen(Error_output,'a');
    fprintf(fid,'&&&&&&&&&& ITERATION %5G\r\n',Iteration);
    fprintf(fid,'%s\r\n',ME.identifier);
    fprintf(fid,'%s\r\n\r\n',ME.message);
    fprintf(fid,'FAILURES:\r\n');
        for e=1:length(ME.stack)
            fprintf(fid,'%s at %i\r\n',...
                ME.stack(e).name,ME.stack(e).line);
        end
    fprintf(fid,'\r\n');
    fprintf(fid,'X value=\r\n');
    fprintf(fid,'%25G\r\n', x0');
    fprintf(fid,'\r\n\r\n');
    fclose(fid);

    ResultData(length(x0)+3,Iteration)= nan;
    ResultData(length(x0)+4:2*length(x0)+3,Iteration)=...
        nan(1,length(x0))';

    ResultData(2*length(x0)+4:2*length(x0)+3+...
        RunOptions.NumConstr,Iteration)=...
            [Geo.Material.MaxStress, ]';
    ResultData(2*length(x0)+4+RunOptions.NumConstr:...
        (length(x0)+1)*(RunOptions.NumConstr+2)+1,...
            Iteration)=nan(1,RunOptions.NumConstr*...
                length(x0))';
    c=ResultData(2*length(x0)+4:2*length(x0)+3+...
        RunOptions.NumConstr,Iteration)';
    ceq=[ ];
    gradcprov= ResultData(2*length(x0)+4+...
        RunOptions.NumConstr:(length(x0)+1)*...
            (RunOptions.NumConstr+2)+1,Iteration)';

    gradc=zeros(length(x0),RunOptions.NumConstr);
    for cons=1:RunOptions.NumConstr
        gradc(1:length(x0),cons)=gradcprov(1,1+(cons-...
            1)*length(x0):cons*length(x0))';
    end
    gradceq=[ ];
```

El análisis MEF debe dar salida a *ceq*, *gradceq* (aunque sea un vector vacío) y la información de *gradc* ha de ser procesada de la misma forma que antes.

```
        ResultData((length(x0)+1)*(RunOptions.NumConstr+2)+2,...
            Iteration)= nan;
        ResultData((length(x0)+1)*(RunOptions.NumConstr+2)+3,...
            Iteration)= nan;

        ResultData((length(x0)+1)*(RunOptions.NumConstr+2)+4,...
            Iteration)= nan;
        ResultData((length(x0)+1)*(RunOptions.NumConstr+2)+5,...
            Iteration)= nan;

        tEndAna= toc(tStart);
        tEndIter= tEndAna;

        ResultData((length(x0)+1)*(RunOptions.NumConstr+2)+6,...
            Iteration)= tEndAna;
        ResultData((length(x0)+1)*(RunOptions.NumConstr+2)+7,...
            Iteration)= tEndIter;

    end

end

if (ResultData(2*length(x0)+4:2*length(x0)+3+...
    RunOptions.NumConstr,Iteration)<=10*eps*...
        ones(RunOptions.NumConstr,1)) &&...
            (ResultData(1,Iteration)==RunOptions.ResFunFEM)

    close all;

    DMesh(-1,1,0,0);
    DGeo(0,0,0);

    saveas(gcf,fullfile(ResultsPath,'Detailed Acceptable...
        Geometries',strcat(['Geometry' ' ' num2str(Iteration)...
            ' Detailed' '.jpg'])));

    axis([RunOptions.ScaledAxisXmin...
        RunOptions.ScaledAxisXmax...
            RunOptions.ScaledAxisYmin...
                RunOptions.ScaledAxisYmax]);
    saveas(gcf,fullfile(ResultsPath, 'Scaled Acceptable...
        Geometries',strcat(['Geometry' ' ' num2str(Iteration)...
            ' Scaled' '.jpg'])));

end
```

```
Data_output = fullfile(ResultsPath,'Result Data.txt');
fid = fopen(Data_output,'a');
fprintf(fid,'&&&&&&&&&& ITERATION=\r\n');
fprintf(fid,'%25G\r\n\r\n', Iteration);
fprintf(fid,'Iteration Origin=\r\n');
fprintf(fid,'%25G\r\n\r\n', ResultData(1,Iteration));
fprintf(fid,'X=\r\n');
fprintf(fid,'%25G\r\n', ResultData(3:length(x0)+2,Iteration));
fprintf(fid,'\r\n');
fprintf(fid,'Volume=\r\n');
fprintf(fid,'%25G\r\n\r\n', ResultData(length(x0)+3,Iteration));
fprintf(fid,'Constraints=\r\n');
fprintf(fid,'%25G\r\n', ResultData(2*length(x0)+4:2*length(x0)+...
    3+RunOptions.NumConstr,Iteration));
fprintf(fid,'\r\n');
fprintf(fid,'Mesh Error (%%)=\r\n');
fprintf(fid,'%25G\r\n\r\n', ResultData((length(x0)+1)*...
    (RunOptions.NumConstr+2)+3,Iteration));
fprintf(fid,'Iteration Time (sec)=\r\n');
fprintf(fid,'%25G\r\n\r\n\r\n', ResultData((length(x0)+1)*...
    (RunOptions.NumConstr+2)+7,Iteration));
fclose(fid);

if numNNetUses>=RunOptions.maxNNetUses
    UseNNet=0;
end

if numFEMUses>=RunOptions.maxFEMUses
    thereIsNNet= 0;
    numNNetUses= 0;
    numFEMUses= 0;
end

save(fullfile(ProblemPath,'Stored ResultData',...
    strcat(['ResultData' ' ' num2str(ActualDate)...
        '.mat'])),'ResultData');
```

Aquí termina el código del módulo *RestrictionFunction.m*, compuesto por los mismos bloques que el módulo anterior con un tratamiento de los datos ligeramente distinto.

B.2.4. *CreateNNetDataPool.m*

Éste es el módulo que define el contenido de la variable *NNetDataPool*. Está presente en el directorio principal del programa.

En el momento de la redacción de este manual, sólo hay implementada una estrategia.

A continuación se presenta y explica el código en detalle:

```
function NNetDataPool= CreateNNetDataPool(mode)

global RunOptions Iteration PrevResData ResultData

switch mode

    case 1
        if RunOptions.UsePrevData==1 && Iteration>1
            NNetDataPool= [PrevResData,ResultData(:,...
                1:Iteration-1)];
        elseif RunOptions.UsePrevData==1 && Iteration==1
            NNetDataPool= PrevResData;
        elseif Iteration>1
            NNetDataPool= ResultData(:,1:Iteration-1);
        else
            NNetDataPool= [ ];
        end

end
```

Se declaran las variables globales necesarias para el correcto funcionamiento del módulo y se presenta una lista de estrategias a la hora de definir el contenido de la variable *NNetDataPool*.

La única implementada consiste en la utilización conjunta de todos los datos de las variables *ResultData* y *PrevResData*.

B.2.5. *ProblemData.m*

Éste es el módulo en el que el usuario define los parámetros de una ejecución determinada del programa. Está presente en el directorio de cada problema.

Para la redacción de este subapartado se ha elegido el *ProblemData.m* de un problema de cuatro variables de diseño. A continuación se presenta y explica el código en detalle:

```
function ProblemData(a)

global IniValRes RunOptions SensAnaIter SensAnaGlob Geo...
    Reference BatchInfo PlaneStrain Quadratic

switch a

    case 'IniValRes'

        IniValRes.x0=[20 14 14 20];
        IniValRes.Aineq=[-1.0 0.0 1.0 0.0; 0.0 1.0 0.0 -1.0];
        IniValRes.bineq=[-0.5; -0.5];
        IniValRes.Aeq=[ ];
        IniValRes.beq=[ ];
        IniValRes.lb=[10 7 7 10];
        IniValRes.ub=[25.0 25.0 25.0 25.0];
```

Se define la función *ProblemData.m* y se declaran las variables de diseño necesarias para transmitir a los demás módulos los parámetros necesarios para su correcto funcionamiento.

Se definen los valores iniciales de las variables de diseño, así como las restricciones de desigualdad entre ellas (que el valor de una variable no supere al de otra) y las de igualdad (si las hubiera), y los límites inferior y superior de dichos valores.

```
case 'RunOptions'

        RunOptions.NumTrialPoints= 20;
        RunOptions.NumStageOnePoints= 20;

        RunOptions.Algorithm= 'sqp';
        RunOptions.Display= 'off';
```

```
RunOptions.TolFun= 10^(-3);
RunOptions.TolX= 10^(-3);
RunOptions.UseParallel= false;
```

Se definen varias opciones del proceso de optimización: el número de puntos de prueba a partir de los cuales el programa intentará encontrar mínimos locales, el algoritmo utilizado en esa tarea, el modo de presentación de resultados, las tolerancias de las que se valdrá el programa para saber que ha llegado a la solución, y la utilización de la computación en paralelo.

```
RunOptions.UsePrevData= 0;

RunOptions.UseNNet= 0;
```

El usuario decide si desea utilizar datos almacenados de ejecuciones anteriores del mismo problema, así como el uso de las redes neuronales en la realización de cálculos.

```
RunOptions.DataPoolOrigin= 1;

RunOptions.VolNNetnumNeurons= 20;
RunOptions.ConstrNNetnumNeurons= 20;

RunOptions.NNetTrainFcn= 'trainlm';
RunOptions.NNetminInputs= 20;
RunOptions.NNetnumLoop= 4;
RunOptions.maxNNetUses= 5;
RunOptions.maxFEMUses= 5;
```

Se definen parámetros relacionados con las redes neuronales: la estrategia a la hora de construir la variable *`NNetDataPool`* (en el momento de la redacción de este manual, sólo hay una implementada), el número de neuronas con las que se construirán las redes neuronales del volumen y las restricciones, la función que se usará para el entrenamiento de las redes neuronales, el número mínimo de datos provenientes de análisis MEF diferentes que se requiere para la creación de las redes, el número de redes que se crearán en un bucle para la elección de la que tenga menor error de validación, el número máximo de usos consecutivos de unas redes determinadas, y el número máximo de

análisis MEF entre el fin de la utilización de un conjunto de redes y la construcción del siguiente.

```
RunOptions.NumConstr= 1;
```

El usuario define el número de restricciones que tiene en cuenta el proceso de optimización para alcanzar la solución. Debe coincidir con las dimensiones de los vectores correspondientes en los otros módulos.

```
RunOptions.ScaledAxisXmin= 0;
RunOptions.ScaledAxisXmax= max(IniValRes.ub)+...
    min(IniValRes.lb);
RunOptions.ScaledAxisYmin= 0;
RunOptions.ScaledAxisYmax= max(IniValRes.ub)+...
    min(IniValRes.lb);
```

Se definen los ejes fijos que se utilizarán para representar las geometrías correspondientes. El usuario puede darles el valor deseado; el existente es sólo una sugerencia.

```
case 'StartGeo'

    load('Geometria_NurbsOpt_4var_squared.mat');

    Geo.Cv(1).FixedCv=0;
    Geo.Cv(2).FixedCv=0;
    Geo.Cv(3).FixedCv=0;

    Geo.Material.MaxStress= 2e6;
    Geo.Material.ElastModul= 10100000;
    Geo.Material.StressFOS= 1;
```

Se carga la geometría definida por el usuario, se indican las curvas de ésta que no son fijas, y se definen las propiedades del material (tensión máxima admisible, módulo de elasticidad y coeficiente de seguridad de las tensiones).

```
    case 'ProcessGeo'

        Geo.Pt.XYZ(2,4)=Reference.Parameters(4);
        Geo.Pt.XYZ(1,6)=Reference.Parameters(2);
        Geo.Pt.XYZ(2,6)=Reference.Parameters(3);
        Geo.Pt.XYZ(1,3)=Reference.Parameters(1);

        SensAnaGlob.Param.DesVarFile=...
            'sa_SetDesVar_NurbsOpt_4var.m';
```

En el proceso de análisis MEF, se asignan las coordenadas de determinados puntos de la geometría a las variables de diseño correspondientes según lo definido aquí por el usuario.

Se establece la función encargada de modificar la geometría con las variables de diseño.

```
    case 'MeshParam'

        BatchInfo.HadaptiveMethod=1;
        BatchInfo.AnalysisType='h_adaptive_mesh';
        BatchInfo.EType=2;
        BatchInfo.IniMesh=5;
        BatchInfo.IntsMesh=5;
        BatchInfo.MaxLevelMesh=11;
        BatchInfo.SquareMesh=1;
        BatchInfo.UniformRefinement=0;
        BatchInfo.RecovType=300;
        BatchInfo.TargErr=[-1 -1 -1 -1 1 -1];
        BatchInfo.ErrReduct=50;
        BatchInfo.MxRefInc=3;
        BatchInfo.MxRefLevel=11;
        BatchInfo.MxNIter=3;
        BatchInfo.GeometricalRefinamentArea=0;
        BatchInfo.GeometricalRefinamentCurvature=1;
        BatchInfo.ExactPressSolution=0;
        BatchInfo.AnalysisMode2D=PlaneStrain;
        BatchInfo.LoadCase2Solve=[1 2 3 4 5 6];
        BatchInfo.Error_Control=0;
        BatchInfo.ParTrans=Quadratic;
        BatchInfo.TerrorTerm=0;
        BatchInfo.MeshCoords=[-1.99 30;-1.99 30];

end
end
```

Se definen todos los parámetros necesarios para el proceso de mallado del bloque de análisis MEF.

El tipo de elemento (*EType*) puede ser lineal (1) o cuadrático (2).

El máximo error en norma energética deseado en la malla se indica en el campo *TargErr* (los valores -1 indican que esos casos de carga no sirven para refinar la malla; los primeros están relacionados con los campos de velocidades de las cuatro variables de diseño, y el último con las sensibilidades).

El máximo número de iteraciones de la malla en el proceso h-adaptativo se indica en *MxNIter*.

Las coordenadas de la malla son definidas por el usuario en *MeshCoords*. La malla debe cubrir toda la geometría.