Proyecto Fin de Grado
Grado en Ingeniería Aeroespacial

Implementación eficiente en MATLAB del Método del Residuo Equilibrado.

Autor: Antonio Jesús Camúñez Delgado

Tutor: Juan Bosco García Archilla

Dpto. Matemática Aplicada II
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2021

Proyecto Fin de Grado
Grado en Ingeniería Aeroespacial

Implementación eficiente en MATLAB del Método del Residuo Equilibrado.

Autor:
Antonio Jesús Camúñez Delgado

Tutor:
Juan Bosco García Archilla
Catedrático

Dpto. Matemática Aplicada II
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2021

Proyecto Fin de Grado: Implementación eficiente en MATLAB del Método del Residuo Equilibrado.

Autor: Antonio Jesús Camúñez Delgado
Tutor: Juan Bosco García Archilla

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Agradecimientos

En primer lugar, a mi tutor por su gran profesionalidad y sinceridad desde el primer días hasta el último.
A mi familia por confiar ciegamente en cada una de mis decisiones y ayudarme en todo lo posible.
A Susana porque sin su apoyo y su forma de hacerme pensar no podría haber acabado este proyecto

Antonio J. Camúñez Delgado

Sevilla, 2021

Resumen

El Método de lo Elementos Finitos es una herramienta muy extendida en el campo de la ingeniería para encontrar soluciones aproximadas a problemas con ecuaciones en derivadas parciales, que no suelen tener solución analítica. Hay diversas formar de estimar el error de los resultados calculados, uno de ellos es el Método del Residuo Equilibrado.

Tras exponer las bases teóricas, se utilizará MATLAB para elaborar un código que integre el Método del Residuo Equilibrado. Siendo la finalidad del proyecto llegar a elaborar un código definitivo cuyos tiempos tiempos de cálculo sean prácticos, demostrando que se puede conseguir una implementación en MATLAB. Para conseguirlo se estudiarán qué factores ralentizan los cálculos y se irán reduciendo su influencia al mínimo posible.

Abstract

Nowadays, Finite Elements Method is a such a universal analytic tool for engeneers due to the amount of complex porblems that envolve Partial differential equations that they have to face. It gives an aproximate solution to those problems. Nevertheless, it is required other tools to measure the error, one of them is the Equilibrated Residual Method. MATLAB is going to be used in order to integrate this procedure into computations. The porpuse behind elaborating this proyect is proving that it can be used in this computer language in a practical way. To archive this goal, there are going to be some analysis to find the key factors that make calculation time increase. After every of them, the MATLAB code is going to get updated with the intention of having a practical tool using MATLAB.

Índice Abreviado

Índice

Introducción

En la ingeniería existen muchos métodos analíticos que permiten obtener e forma directa el resultado exacto de un problema. El inconveniente de éstos es que no todos los problemas se pueden resolver se esta forma, ya que, esas soluciones analíticas están asociadas a un tipo de problema con unas dimensiones y unas condiciones de contorno en concreto, como por ejemplo la flecha de una viga empotrada en un extremo.

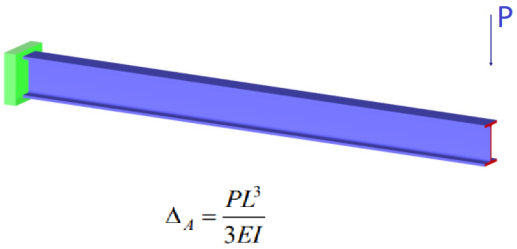

Figura 0.1 Viga en voladizo.

En los casos prácticos con los que se encuentra un ingeniero casi siempre hay envueltas ecuaciones en derivadas parciales, con cierta complejidad, aplicadas sobre un amplio abanico de situaciones. Luego, no hay una solución analítica para cada una de ellas. Para obtener una aproximación de la solución se recurre a métodos numéricos, que utilizan el cálculo computacional para obtenerla. De la mano de la solución aproximada nace el concepto de error, que será la diferencia entre la solución numérica y la real.

Uno de estos procedimientos más usados sobre muchos campos de la ingeriería es el Método de los Elementos Finitos. Éste se caaracteriza por dividir el cuerpo estudiado en partes más pequeñas, llamadas elementos, dentro de las que se expresa la solución únicamente como función de unos valores en sus vértices que se utilizan para definir funciones, que generalmente serán lineales o cuadráticas. Es decir, la solución aproximada será la combinación de la de cada elemento, teniendo una forma sencilla dentro de cada uno.

La válidez de esta aproximación vendrá determinada por el valor del error. Ahí es donde entran el juego los métodos de estimación del error *a posteriori*, que reutlizan datos calculados durante la obtención de la solución aproximada con elementos finitos. El Método del Residuo Equilibrado es uno de ellos, con el que se obtiene una estimación de la aportación al error de cada elemento gracias a plantear al problema de una forma en concreto. Al conocer la validez de la solución en cada elemento se puede decidir si es necesario dividir algunos en partes más pequeñas, con el fin de llegar a una solución que se

acerque lo suficiente a la exacta.

El tiempo de cálculo es un factor crucial al usar estas herramientas, estando altamente relacionados con la productividad de un ingeniero en su jornada laboral. De modo que tanto los procesos para obtener la aproximación y comprobar su validez deben de estar optimizados, de tal forma que sean lo más rápido posible. Este proyecto estará centrado en implementar y optimizar el Método del Eesiduo Equilibrado utilizando utilizando programa de cálculo computacional MATLAB.

1 Estructura del proyecto

En este capítulo se expondrá la disposición de contenido que se tendrá a lo largo de este proyecto. En primer lugar, se hará una descripción sobre la base teórica del Método de los Elementos Finitos en la que de destacarán sus principales ideas características. Además se verá el problema que será considerado en todo el proyecto. Después, se explicará la base teórica del Método del Residuo Equilibrado, viendo cada una de las elecciones que se hacen para conseguir expresar la estimación del error como la superposición de la influencia de cada elemento. Con todo esto, se podrá generar un primer código de MATLAB que implemente cada uno de los pasos del Método del Residuo Equilibrado con el fin de ser una base sobre la que trabajar. Por último se estudiará el rendimiento del mismo y se irán realizando diferentes modificaciones que hasta llegar a un código definitivo, consecuencia del proceso de optimización.

2 Método de los Elementos Finitos

En esta sección se hará una descripción del Método de los Elementos Finitos, que será el utilizado a la hora de calcular la solución aproximada, cuyo error se estimará con el Método del Residuo Equilibrado.

2.1 Introducción

Como se nombró anteriormente, a la hora de resolver problemas se encuentran diversas ecuaciones algebráicas, intregrales, diferenciales o en derivadas parciales que no pueden ser resueltas de forma directa. Así, es necesario a recurrir a métodos numéricos para obtener una solución aproximada. A lo largo de la historia se han usado algunos como el Método de las Diferencias Finitas, con el que se discretiza el domino y obtiene una aproximación mediante la linealización de las ecuaciones diferenciales del problema en cada una de sus partes. De modo que se transforma una ecuación diferencial en otra algebraica. Otra opción utilizada por el Método de Galerkin y el de los Mínimos cuadrados fue la de obtener una solución como la combinación lineal de funciones, $c_j\psi_j$, estableciendo las funciones aproximantes, ψ_j, y siendo c_j las incognitas. [1]

En base a esta última idea surge el Método de los Elementos Finitos, a mediados del siglo XX en la aplicación de análisis estructural de sistemas de barras. Con la particularidad de que, además de hacer una discretización del dominio, se hace la elección de las funciones aproximantes unas definidas a trozos, conocidas como funciones de pequeño soporte. Su uso se extrapoló a otros muchos campos como la Tranferencia de Calor, el Electromagnetismo o la Mecánica de Fluidos.[1]

2.2 Base teórica

Inicialmente, el problema se define con la búsqueda de $u(x)$, para cada cada x existente en el dominio Ω, que satisface:

$$\left.\begin{array}{rcccl} \mathcal{D}(u) & = & f & \text{para} & x \in \Omega \\ u & = & u_D & \text{para} & x \in \Gamma_D \\ \dfrac{\partial u}{\partial n} & = & g & \text{para} & x \in \Gamma_N \end{array}\right\}$$

donde:

- $\mathcal{D}(u) = f$, ecuación diferencial que se tiene que cumplir en todos los puntos.

- Γ_D, frontera Dirichlet, en la que se conoce el valor de la incógnita.
- Γ_N, frontera Neumann, en la que el valor del flujo de u vendrá dada por la función g.
- $\partial\Omega$, contorno del dominio. Se cumple que $\partial\Omega = \Gamma_D \bigcup \Gamma_N$.

Para poder obtener una solución aproximada mediante el Método de los Elementos Finitos es necesario reformular el problema en formulación débil, es decir, se escriben de forma integral y las soluciones soluciones pertenecen a un espacio de Banach V, es decir, es un espacio normado y completo, que en resumidas cuentas se puede expresar como que cada vector contenido en el mismo tiene norma finita y que cualquier diferencia entre dos de sus vectores está en V. Quedando el problema expresado de la siguiente forma:

$$u \in V \quad \text{tal que} \quad B(u,v) = L(v) \quad ,\forall v \in V$$

donde:

- V, espacio de soluciones.
- $B(u,v)$, función que toma como entrada dos vectores de una misma longitud, con los que se obtiene otro con las mismas dimensiones.
- $L(v)$, función que transforma un vector en otro de su misma dimensión.[3] [2]

El siguiente paso en el MEF es relizar la discretización del dominio, $\mathcal{P}$, es decir, se dividirá el dominio Ω en un cierto número de partes, conocidas también como elementos, con formas definidas por líneas rectas, quedando unos vértices en cada uno de ellos que se conocen como nodos. Tras llevar a cabo este proceso se tiene un mallado, cuyas principales características son tener un número n_T de elementos y n_N nodos. Si se tiene un problema unidimensional, cada elemento será un segmento definido por los nodos de sus extremos. Mientras que en los problemas bidimensionales, los elementos suelen ser triángulos, con 3 nodos, o cuadriláteros, con 4 nodos.

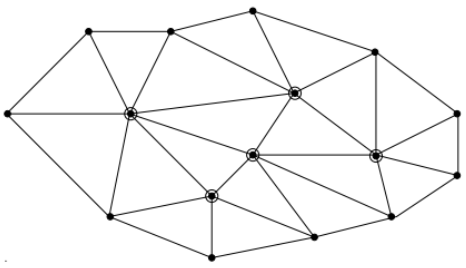

Figura 2.1 Ejemplo de una discretización con elementos triangulares. [2].

Una vez se tienen ésto, se puede definir la solución aproximada, u_x, de igual forma que en el Método de Galerkin y el de los Mínimos cuadrados. La aproximación será la combinación lineal entre varias funciones, quedando definida por la elección de las funciones ψ_i, y los coeficientes c_i, que están por determinar. De tal forma que:

$$u_x(x) = \sum_{i=1}^{N} c_i \psi_i(x)$$

La gran ventaja del MEF nace con la elección de estas funciones aproximantes psi_i, ya que, se toman funciones definidas a trozos asociadas a cada nodo, conocidas como funciones de pequeño soporte. De forma que son 1 en el nodo y 0 en en los otros nodos, es decir, únicamente serán no nulas en los elementos colindantes al nodo al que están

asociadas. En la práctica, estas funciones son desde lineales hasta cúbicas. En consecuencia a esta elección, el valor de los coeficientes asociados a cada función será el valor de la solución aproximada en el nodo al que están asociadas. Las funciones de pequeño soporte se suelen ser simples, siendo lineales, cuadráticas o cúbicas. [1]

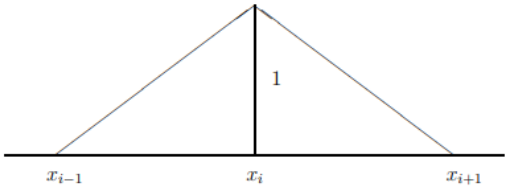

Figura 2.2 Función de pequeño soporte lineal en caso unidimendional.[2].

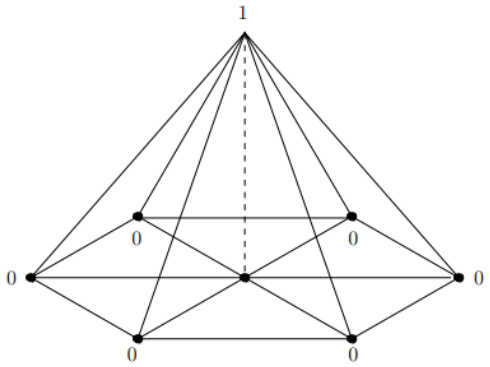

Figura 2.3 Función de pequeño soporte lineal en caso bidimendional.[2].

Incluyendo
esto a la formulación del problema se llega a encontrar U_i que cumpla lo siguiente:

$$\sum_{i=1}^{n_N} c_i B(\psi_i,\psi_j) \quad = \quad l(\psi_j) \quad , j = 1,2...n_N$$

Es decir, se tiene un sistema de ecuaciones lineales de n_N incógnitas con n_N ecuaciones, que puede ser representada de forma simplificada como $K\,u = f$. En la que K se conoce como matriz de rigidez, en la Mecánica de Sólidos Continuos. Gracias a que las funciones son de pequeño soporte, el término que multiplica a cada c_i será no nulo cuando $i = j$ o cuando i y j estén en un mismo elemento. Consecuentemente la matriz K será una matriz con una cantidad importante de elementos nulos, lo que hace que sea muy ventajoso definirla como una matriz dispersa. [2]

Una vez se calculan esos coeficientes ya se tienen definidas todas las variables necesarias para conocer el valor de la solución aproximada en cada punto del dominio.

2.3 Problema considerado

A la hora de trabajar en la optimización del funcionamiento del código del Método del Residuo Equilibrado se fijará un problema, que será resuelto mediante el MEF para varios mallados del dominio.

El problema será bidimensional con una solución de una función escalar, por lo que

se podría asimilar a un problema de transmisión de calor en una placa, en el que la incógnita es el valor de la temperatura. El dominio de este problema, Ω, será un cuadrado de lado de longitud 1 en el que se tiene la siguiente ecuación en derivadas parciales y condiciones de contorno.

$$\Omega = [0,1] \times [0,1]$$
$$-\nabla^2 u + u = f \text{ en } \Omega$$
$$\frac{\partial u}{\partial n} = g \text{ en } \Gamma_N$$
$$u = 0 \text{ en } \Gamma_D$$

siendo

$$\nabla^2 u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\check{z}\partial y^2}$$
$$f = 16\,x\,(1-x) + 16\,y\,(1-y) + 8\,x\,(1-x)\,y\,(1-y)$$
$$g = \begin{cases} -8\,y\,(1-y) & si \quad x = 0 \text{ o } x = 1 \\ -8\,x\,(1-x) & si \quad y = 0 \text{ o } y = 1 \end{cases}$$

Este problema se solucionará con un programa de MATLAB dado, utilizando elementos triangulares y con funciones de pequeño soporte lineales. El mallado el número de elementos del mallado estará definido por el parámetro N, número de nodos que hay en cada lado de la frontera. Además se generarán de dos formas, una en la que todos los triángulos tienen el mismo tamaño, y otra en la que sus dimensiones son aleatorias, asimilándose más a las discretizaciones que se dan en la práctica, en la que el mallado es más fino en unas zonas que en otras. A continuación se expone un ejemplo de cada tipo, con su correspondiente solución.

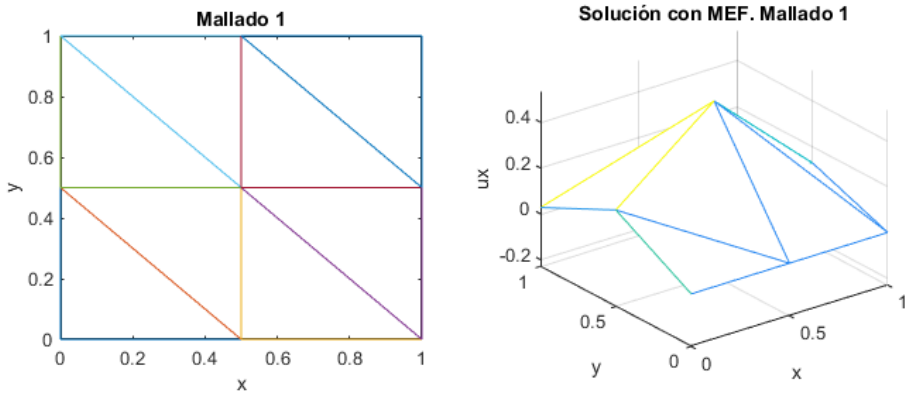

Figura 2.4 Mallado regular de 8 elementos.

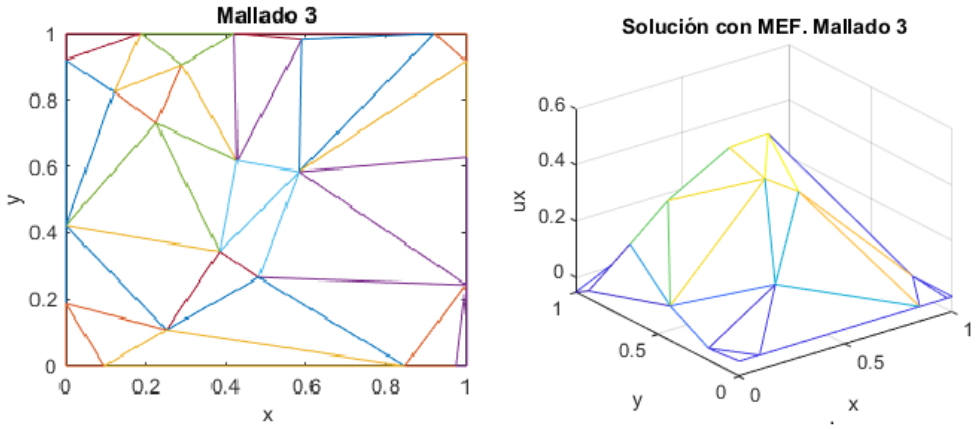

Figura 2.5 Mallado regular de 32 elementos.

3 Método del Residuo Equilibrado

En este apartado se expondrá teóricamente el Método del Residuo Equilibrado.

3.1 Introducción

Al igual que en todos los métodos numéricos, en el Método de los Elementos Finitos hay una diferencia entre la solución exacta y la aproximada, generalmente llamada error. Este valor tiene una gran importancia, ya que, determinará la validez del resultado. Luego es necesario conocerlo en la práctica para tener la certeza de que los cálculos que se han obtenido son válidos para ser interpretados. Es de aquí donde surgen los métodos para estimar el error de la solución, para saber si el mallado que se ha creado es suficientemente refinado. Es decir, obtener una solución definitiva implica generar un mallado, obtener la solución y comprobar si el error está dentro de los valores admisibles, en caso negativo se debe de repetir con otro mallado.

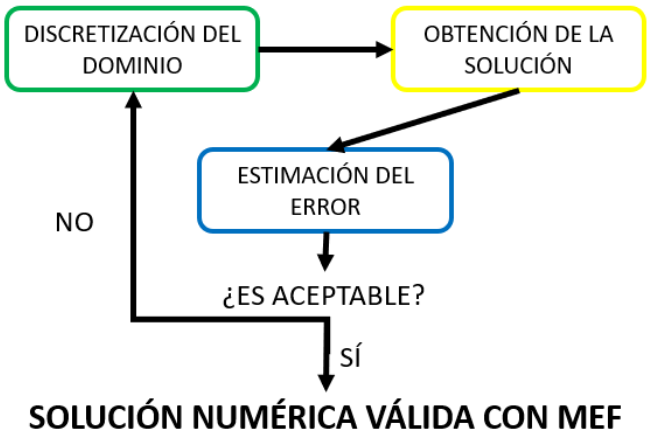

Figura 3.1 Esquema del procedimiento para obtener una solución aceptable.

En el caso de que se tenga una división regular de todo el dominio y se conozca el valor máximo del error, el refinamiento del mallado es sencillo, ya que, sólo hay que aumentar el número de elementos. Sin embargo, este procedimiento no es el óptimo si se quiere tener un funcionamiento óptimo de los programas que se usan en el cálculo debido a que se puede estar aumentando el tiempo de calculo por tener más elementos. Por consiguiente, no sólo es interesante conocer la estimación del error máximo en todo el

dominio, sino que también la aportación al mismo de cada elemento. De modo que se introduzcan más elementos sólo en los lugares en los que sean necesarios. A este proceso se le conoce como Elementos Finitos adaptados, cuya propiedad fundamental es encontrar un mallado óptimo para un problema y un error dado.

Para poder desarrollar estos algoritmos son necesarios métodos de estimación del error *a posteriori*, que utilizan datos del propio cálculo de Elementos Finitos para resolver de forma aproximada la ecuación que define al error. Éstos empezaron a desarrollarse durante la década de los años 1980 para conseguir establecerse en la siguiente década. De aquí surgen dos grupos, los estimadores de proyección de flujo y los estimadores residuales, entre los cuales se encuentra el Método del Residuo Equilibrado. [4]

3.2 Descripción del método

como se ha visto en la sección anterior, la solución exacta del problema ,u, será la que satisface la siguiente ecuación para todos los puntos del dominio.

$$B(u,v) = L(v)\ \forall v \in V \tag{3.1}$$

Sin embargo, la solución de elementos finitos, u_x, no es igual a u, existiendo un error ,e, definido como la diferencias de ambas soluciones. Por lo que se puede expresar $B(e,v)$ de la siguiente forma:

$$B(e,v) = B(u,v) - B(u_x,v) = L(v) - B(u_x,v)\ \forall v \in V \tag{3.2}$$

De aquí, la norma del error se obtiene como

$$\|e\| = \sup_{v\in V,\, v\neq 0} \frac{|B(e,v)|}{\|v\|} = \sup_{v\in V,\, v\neq 0} \frac{|L(v) - B(u_x,v)|}{\|v\|} \tag{3.3}$$

[4]

Para poder expresar su valor para cada elemento habrá que expresar $L(v) - B(u_x,v)$ como uno o varios sumatorios con la contribución de cada elemento. Ésto se efectuará obteniendo la solución local de cada uno de ellos, formulando cada uno independientemente como un problema de contorno con frontera Neumann en todos sus lados. Es decir, tiene que asignar un flujo en cada una las fronteras de los elementos, dependiendo únicamente de ellos mismos y sus elementos colindantes. Siendo la particularidad de esta herramienta las forma en las que los flujos se definen. [4]

El flujo sobre la frontera del elemento K, g_K, se aproxima como

$$g_K \approx n_K \cdot \nabla u|_K \quad \text{sobre} \quad \partial K \tag{3.4}$$

siendo n_K la normal exterior al elemento K y ∂K su frontera. [4]

Los flujos tendrán que ser continuos a través de la frontera de un elemento con su elemento vecino, es decir,

$$n_K \cdot \nabla u|_K + n_{K'} \cdot \nabla u|_{K'} = 0 \quad \text{sobre} \quad \partial K \cap \partial K' \ . \tag{3.5}$$

En el caso de los lados que sean parte de la frontera Neumann del problema completo, el valor del flujo será igual al que establece la función g, con la finalidad de impedir que se imponga un flujo aproximado en los lados en los que se conoce el flujo exacto.

$$g_K = g \quad \text{sobre} \quad \partial K \cap \Gamma_N \ . \tag{3.6}$$

Con ambas imposiciones se puede afirmar que para todo $v \in V$,

$$\int_{\Gamma_N} gv\,ds = \sum_{K=1}^{n_T} \int_{\partial K} g_K v\,ds. \tag{3.7}$$

siendo n_T el número de elementos que componen la discretización.

Se puede comprobar que esto se cumple , ya que los lados interiores elementos la aportación de un elemento se cancela con la de su veceno en ese lado, según establece (3.5). Para el caso de la frontera Dirichlet, al estar impuesta u, v será nula en todos los lados de ese tipo, por definición. [4]

El error residual se puede expresar elemento a elemento de la siguiente forma,

$$L(v) - B(u_x,v) = \sum_{K=1}^{n_T} \{(f,v)_K - B_K(u_x,v)\} + \int_{\Gamma_N} gv\,ds \tag{3.8}$$

para cada $v \in V$, donde

$$B_K(u,v) = \int_K (\nabla u \cdot \nabla v + uv)\,dx \tag{3.9}$$

y

$$(f,v)_K = \int_K fv\,dx. \tag{3.10}$$

siendo

$$L(v) = \sum_{K=1}^{n_T} (f,v)_K + \int_{\Gamma_N} gv\,ds$$

Luego se puede escribir como la suma de un término de cada elemento de esta forma

$$L(v) - B(u_x,v) = \sum_{K=1}^{n_T} \left\{ (f,v)_K - B_K(u_x,v) + \int_{\partial K} g_K v\,ds \right\} \tag{3.11}$$

El término entre corchetes será la aportación de cada elemento, cuyo valor no es conocido. Es decir, éste se puede expresar como

$$B_K(\phi_K,v) \quad = \quad (f,v)_K - B_K(u_x,v) + \textstyle\int_{\partial K} g_K v\,ds \quad \forall v \in V$$

siendo ϕ_k la solución del problema del residuo local.[4]

Aprovechando esta formulación, y que se cumple para todo v, se fija la restricción de que $B_K(\phi_K,1) = 0$. Ésto implica que debe de existir un equilibrio entre los flujos en los bordes y los que se producen en su interior, debido a que cuando $v = 1$ sólo se tienen en cuenta las variables reales del problema. Esta restricción sería equivalente a imponer un balance energético en un problema de Transferencia de Calor, es de aquí de donde surge el nombre del método. [4]

Todas estas condiciones que deben cumplir los flujos aproximados en cada elemento se pueden resumir como

$$\left.\begin{aligned} (f,1)_K - B_K(u_x,1) + \int_{\partial K} g_K\,ds &= 0 \\ g_K + g_{K'} &= 0 \quad \text{sobre} \quad \partial K \cap \partial K' \\ g_K &= g \quad \text{sobre} \quad \partial K \cap \Gamma_N \end{aligned}\right\} \tag{3.12}$$

Estas condiciones también se pueden expresar para que se cumpla en cada nodo del elemento, utilizando las funciones de la base de Lagrange del espacio X, θ_n de cada uno. Se cumple que

$$\sum_{n\in\mathcal{N}(K)} \theta_n(x) = 1 \quad \text{en} \quad K \tag{3.13}$$

y consecuentemente

$$\sum_{n\in\mathcal{N}(\gamma)} \theta_n(x) = 1 \quad \text{sobre} \quad \gamma \tag{3.14}$$

donde $\mathcal{N}$ son todos los nodos, $\mathcal{N}(K)$ son los nodos del elemento K y $\mathcal{N}(\gamma)$ son los nodos del lado γ. Se llega a que

$$\left.\begin{aligned} (f,\theta_n)_K - B_K(u_x,\theta_n) + \int_{\partial K} g_K\theta_n\,ds &= 0 \quad \forall n\in\mathcal{N}(K) \\ g_K + g_{K'} &= 0 \qquad \text{sobre} \qquad \partial K\cap\partial K' \\ g_K &= g \qquad \text{sobre} \qquad \partial K\cap\Gamma_N \end{aligned}\right\} \tag{3.15}$$

La forma de los flujos queda complemente determinada en los lados que pertenecen a la frontera Neumann. Sin embargo, en el resto de lados puede existir una cierta libertad a la hora de elegir como serán los flujos. Es decir, puede posible encontrar más de una función que cumpla las condiciones que se cumplen. Aquí es cuando se realiza una decisión clave, se define los flujos como funciones lineales, claramente excepto en los lados en los que es conocido. Con esta elección sólo se tendrán que determinar dos coeficientes para construir los flujos aproximados. Estos dos grados de libertad en cada lado serán μ_K^γlos momentos del flujo ponderados con las funciones base sobre el lado en cuestión.

$$\mu_{K,n}^\gamma = \int_\gamma g_K\theta_n\,ds, \quad n\in\mathcal{N}(\gamma)\ . \tag{3.16}$$

Suponemos que l y r son los nodos de los extremos del lado γ, entonces la expresión del flujo sería

$$g_K|_\gamma = \alpha_l\theta_l + \alpha_r\theta_r$$

donde α_l y α_r son valores que están determinados por los momentos del flujo. De forma que los si se incluye en (3.16) se llega a una expresión que relacionan los momentos y las constantes por determinar.

$$\frac{h}{6}\begin{bmatrix}2 & 1\\ 1 & 2\end{bmatrix}\begin{bmatrix}\alpha_l\\ \alpha_r\end{bmatrix} = \begin{bmatrix}\mu_{K,l}^\gamma\\ \mu_{K,r}^\gamma\end{bmatrix} \tag{3.17}$$

donde h es la longitud del lado. Despejando en este sistema, las constantes en función de los momentos son:

$$\alpha_l = \frac{2}{h}\left(2\mu_{K,l}^\gamma - \mu_{K,r}^\gamma\right)$$

$$\alpha_r = \frac{2}{h}\left(-\mu_{K,l}^\gamma + 2\mu_{K,r}^\gamma\right)$$

De forma que el flujo se expresa como función de los momentos así:

$$g_K|_\gamma = \frac{2}{h}\left\{\left(2\mu_{K,l}^\gamma - \mu_{K,r}^\gamma\right)\theta_l + \left(-\mu_{K,l}^\gamma + 2\mu_{K,r}^\gamma\right)\theta_r\right\} \tag{3.18}$$

[4]

Consecuentemente, el problema reside en obtener los momentos de los flujos. Así que se introduce esta última definición del flujo en (3.15), quedando de esta forma:

$$\left.\begin{array}{rcll} \sum\limits_{\gamma\subset\partial K} \mu^{\gamma}_{K,n} & = & \Delta_K(\theta_n) & \forall n\in\mathcal{N}(K) \\ \mu^{\gamma}_{K,n}+\mu^{\gamma}_{K',n} & = & 0 & \forall n\in\mathcal{N}(K),\quad \gamma=\partial K\cap\partial K' \\ \mu^{\gamma}_{K,n} & = & \int_{\gamma} g\theta_n\,ds & \forall n\in\mathcal{N}(K),\quad \gamma=\partial K\cap\Gamma_N \end{array}\right\} \tag{3.19}$$

donde

$$\Delta_K(\theta_n) = B_K(u_x,\theta_n) - (f,\theta_n)_K \tag{3.20}$$

Como se observa, la ventaja de tomar los grados de libertad en los momentos de los flujos es que éstos se pueden calcular independientemente en cada nodo.[4]

Inicialmente se tenía que encontrar una función cualquiera para el flujo en cada lado, que podía llegar a depender de todo el problema completo si se planteaba como en 3.15), yendo lado por lado. Pero después de realizar la elección sobre los flujos, el problema pasará a ser determinar los dos coeficientes que definen al flujo mediante la resolución unos sistemas de ecuaciones lineales, independientes para cada nodo. Es decir, se podrá obtener un conjunto de flujos que cumplan las condiciones de equilibrio al resolver varios problemas locales sin tener que resolver todo el problema completo. [4]

A la hora de resolver estos problemas locales de cada nodo, se presentan varias situaciones en función de su localización del nodo , ya que restricciones son diferentes si se encuentra en el interior o en la frontera, dependiendo además del tipo de frontera que sean los lados que llegan al nodo. Para verlos, se supondrá un problema genérico tomando un nodo n y su función base asociada θ_n. Se pueden dar las siguientes situaciones:

1. Vértice interior. El nodo se encuentra completamente rodeado por N elementos. Se muestra en la figura 3.2. Se tienen las siguientes restricciones por las condiciones de (3.19), por el nodo n

$$\left.\begin{array}{rcl} \mu^{\gamma_1}_{1,n}+\mu^{\gamma_2}_{1,n} & = & \Delta_1(\theta_n) \\ \mu^{\gamma_2}_{2,n}+\mu^{\gamma_3}_{2,n} & = & \Delta_2(\theta_n) \\ & \vdots & \\ \mu^{\gamma_N}_{N,n}+\mu^{\gamma_1}_{N,n} & = & \Delta_N(\theta_n) \end{array}\right\}$$

y por los lados interiores

$$\left.\begin{array}{rcl} \mu^{\gamma_1}_{1,n}+\mu^{\gamma_1}_{N,n} & = & 0 \\ \mu^{\gamma_2}_{2,n}+\mu^{\gamma_2}_{1,n} & = & 0 \\ & \vdots & \\ \mu^{\gamma_N}_{N,n}+\mu^{\gamma_N}_{N-1,n} & = & 0) \end{array}\right\}$$

2. Nodo fronterizo. Como su nombre indica indica está en la frontera, es decir, 2 de los lados que llegan a él serán frontera y los otros $N-1$ serán interiores, véase Figura 3.3. Siendo las restricciones por el nodo n

$$\left.\begin{array}{rcl} \mu^{\gamma_1}_{1,n}+\mu^{\gamma_2}_{1,n} & = & \Delta_1(\theta_n) \\ \mu^{\gamma_2}_{2,n}+\mu^{\gamma_3}_{2,n} & = & \Delta_2(\theta_n) \\ & \vdots & \\ \mu^{\gamma_N}_{N,n}+\mu^{\gamma_1}_{N,n} & = & \Delta_N(\theta_n) \end{array}\right\}$$

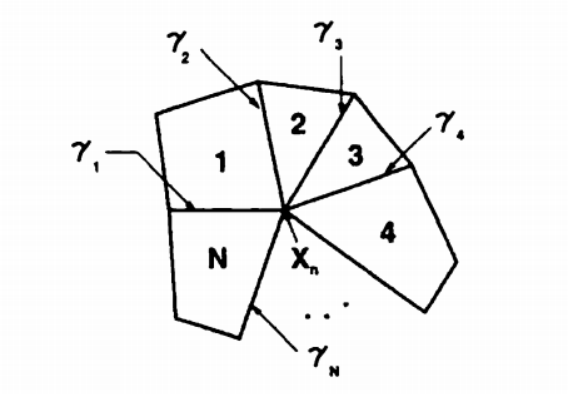

Figura 3.2 Ejemplo de nodo interior. [4].

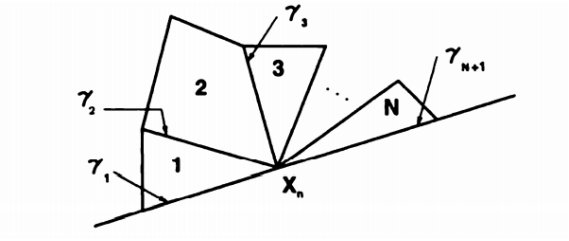

Figura 3.3 Ejemplo de nodo fronterizo. [4].

y por los lados interiores

$$\left.\begin{array}{rcl} \mu_{2,n}^{\gamma_2}+\mu_{1,n}^{\gamma_2} & = & 0 \\ \mu_{3,n}^{\gamma_3}+\mu_{2,n}^{\gamma_3} & = & 0 \\ & \vdots & \\ \mu_{N-1,n}^{\gamma_N}+\mu_{N-1,n}^{\gamma_N} & = & 0 \end{array}\right\}$$

Quedan por determinar 2 restricciones, una en el lado γ_1 y otra en el γ_{N+1}, que dependerán del tipo de frontera que sean los lados que llegan al nodo. Se pueden dar los siguientes tres casos:

2.a Neumann-Neumann. Los lados γ_1 y γ_{N+1} pertenecen a Γ_N, por lo que se tienen las siguientes dos imposiciones para los momentos

$$\begin{array}{rcl} \mu_{1,n}^{\gamma_1} & = & \displaystyle\int_{\gamma_1} g\theta_n\, ds \\ \mu_{N,n}^{\gamma_{N+1}} & = & \displaystyle\int_{\gamma_{1+N}} g\theta_n\, ds \end{array}$$

2.b Neumann-Dirichlet. El lado γ_1 está en la frontera Neumann y γ_{N+1} en la frontera Dirichlet, es válido para el caso de que sean al contrario, ya que sólo se tendrían que intercambiar las restricciones

$$\begin{array}{rcl} \mu_{1,n}^{\gamma_1} & = & \displaystyle\int_{\gamma_1} g\theta_n\, ds \\ \mu_{N,n}^{\gamma_{N+1}} & = & \text{sin restricciones} \end{array}$$

2.c Dirichlet-Dirichlet.os lados γ_1 y γ_{N+1} están contenidos en Γ_D , entonces

$$\begin{aligned} \mu_{1,n}^{\gamma_1} &= \text{sin restricciones} \\ \mu_{N,n}^{\gamma_{N+1}} &= \text{sin restricciones} \end{aligned}$$

Tabla 3.1 Tabla resumen de todos los casos y tipo de solución de cada uno.[4].

	Incógnitas		Restricciones		
Caso	Momentos de flujo	Elementos en equilibrio	Lados interiores	Lados exteriores	Solución
1	$2N$	N	N	0	No única
2.a	$2N$	N	$N-1$	2	Única
2.b	$2N$	N	$N-1$	1	Única
2.c	$2N$	N	$N-1$	0	No única

Como se observa en la tabla, existe solución para todos los casos, aunque para dos de ellos la solución no es única. Es decir, se pueden dar infinitos casos, unos más cercanos en los que la diferencia entre los momentos reales y los aproximados, que son los que se calculan con los sistemas anteriores, puede llegar a variar mucho según la solución que se tome. Así que para conseguir que sean lo más parecidas posibles se tomará la que minimiza la diferencia entre ambas. Luego la función objetivo a minimizar será la siguiente

$$\frac{1}{2}\sum_{K=1}^{n_T}\sum_{\gamma\subset\partial K}\left(\mu_{K,n}^{\gamma}-\tilde{\mu}_{K,n}^{\gamma}\right)^2. \tag{3.21}$$

siendo $\mu_{K,n}^{\gamma}$ los momentos reales y $\tilde{\mu}_{K,n}^{\gamma}$ los momentos aproximados.

$$\mu_{K,n}^{\gamma} = \int_{\gamma} \theta_n n_K \cdot \nabla u \, ds$$

$$\tilde{\mu}_{K,n}^{\gamma} = \int_{\gamma} \theta_n n_K \cdot \nabla u_x \, ds$$

[4]

Para encontrar los óptimos se utiliza la herramienta de los multiplicadores de Lagrange. Quedando la siguiente Lagrangiana

$$\begin{aligned} \mathcal{L}\left(\{\tilde{\mu}_{K,n}^{\gamma}\},\{\lambda_{\gamma}\},\{\sigma_K\}\right) = \frac{1}{2}\sum_{K=1}^{n_T}\sum_{\gamma\subset\partial K}\left(\mu_{K,n}^{\gamma}-\tilde{\mu}_{K,n}^{\gamma}\right)^2 + \sum_{K=1}^{n_T}\left(\Delta_K(\theta_n) - \sum_{\gamma\subset\partial K}\mu_{K,n}^{\gamma}\right) + \\ + \sum_{\gamma=\partial K\cap\partial K'}\lambda_{\gamma,n}\left(\mu_{K,n}^{\gamma}+\mu_{K',n}^{\gamma}\right) + \sum_{\gamma=\partial K\cap\Gamma_N}\lambda_{\gamma,n}\left(\mu_{K,n}^{\gamma}-\int_{\gamma} g\theta_n \, ds\right). \end{aligned}$$

Por comodidad se fija que $\lambda_{\gamma,n}$ sea nulo en los lados de frontera Dirichlet debido a que no hay restricciones en los momentos de esos lados. De forma que a las restricciones de (3.19) se le suma

$$\mu_{K,n}^{\gamma} - \tilde{\mu}_{K,n}^{\gamma} - \sigma_{K,n} + \lambda\gamma,n = 0 \tag{3.22}$$

y

$$\lambda_{\gamma,n} = 0 \quad \text{sobre} \quad \gamma \subset \Gamma_D \tag{3.23}$$

Los multiplicadores de cada lado se puede expresar uniendo estas dos últimas restricciones junto a las de (3.19). Llegando a :

$$\lambda_{\gamma,n} = \begin{cases} \frac{1}{2}\left(\sigma_{K,n} + \sigma_{K',n} + \tilde{\mu}^{\gamma}_{K,n} + \tilde{\mu}^{\gamma}_{K',n}\right) & \gamma = \partial K \cap \partial K \\ \sigma_{K,n} + \tilde{\mu}^{\gamma}_{K,n} - \int_{\gamma} g\theta_n \, ds & \gamma = \partial K \cap \Gamma_N \\ 0 & \gamma = \partial K \cap \Gamma_D \end{cases}$$

Con estas relaciones, se puede expresar los momentos del flujo como función de $\sigma_{K,n}$ y los momentos aproximados, que son conocidos al ser una entrada la solución aproximada de elementos finitos.

$$\mu^{\gamma}_{K,n} = \begin{cases} \frac{1}{2}\left(\sigma_{K,n} - \sigma_{K',n} + \tilde{\mu}^{\gamma}_{K,n} - \tilde{\mu}^{\gamma}_{K',n}\right) & \gamma = \partial K \cap \partial K \\ \int_{\gamma} g\theta_n \, ds & \gamma = \partial K \cap \Gamma_N \\ \sigma_{K,n} + \tilde{\mu}^{\gamma}_{K,n} & \gamma = \partial K \cap \Gamma_D \end{cases}$$

Para poder expresar todas las restricciones sobre los momentos del flujo según el multiplicador de Langrange se incluye esto último en la primera ecuación de (3.19).

$$\frac{1}{2} \sum_{\gamma = \partial K \cap \partial K'} \left(\sigma_{K,n} - \sigma_{K',n}\right) + \sum_{\gamma \subset \partial K \cap \Gamma_D} \sigma_{K,n} = \tilde{\Delta}_K(\theta_n) \; \forall K \in \mathcal{P}_n \tag{3.24}$$

donde

$$\begin{aligned} \tilde{\Delta}_K(\theta_n) = \Delta_K(\theta_n) - \frac{1}{2} \sum_{\gamma = \partial K \cap \partial K'} \left(\tilde{\mu}^{\gamma}_{K,n} - \tilde{\mu}^{\gamma}_{K',n}\right) - \sum_{\gamma \subset \partial K \cap \Gamma_D} \tilde{\mu}^{\gamma}_{K,n} - \sum_{\gamma \subset \partial K \cap \Gamma_N} \int_{\gamma} g\theta_n \, ds = \\ = B_K(u_x, \theta_n) - (f, \theta_n)_K - \int_{\partial K} \left\langle \frac{\partial u_x}{\partial n_K} \right\rangle \theta_n \, ds \end{aligned}$$

siendo

$$\left\langle \frac{\partial u_x}{\partial n_K} \right\rangle = \begin{cases} \frac{1}{2} n_K \cdot \{(\nabla u_x)_K + (\nabla u_x)_{K'}\} & \text{sobre} \quad \partial K \cap \partial K' \\ n_K \cdot (\nabla u_x)_K & \text{sobre} \quad \partial K \cap \Gamma_D \\ g & \text{sobre} \quad \partial K \cap \Gamma_N \end{cases}$$

De esta forma, los multiplicadores de Langrange son las únicas incógnitas calcular para obtener los momentos, y consecuentemente el flujo. Éstos se obtienen de (3.24), llegando a los siguientes sistemas de ecuaciones en cada caso:

1. Nodo interior:

$$\frac{1}{2} \begin{bmatrix} 2 & -1 & & \dots & & -1 \\ -1 & 2 & -1 & \dots & & 0 \\ \vdots & & & & & \vdots \\ 0 & & \dots & -1 & 2 & -1 \\ -1 & & \dots & & -1 & 2 \end{bmatrix} \begin{bmatrix} \sigma_1 \\ \sigma_1 \\ \vdots \\ \sigma_{N-1} \\ \sigma_N \end{bmatrix} = \begin{bmatrix} \tilde{\Delta}_1(\theta_n) \\ \tilde{\Delta}_2(\theta_n) \\ \vdots \\ \tilde{\Delta}_{N-1}(\theta_n) \\ \tilde{\Delta}_N(\theta_n) \end{bmatrix}$$

2. Nodo fronterizo. Se dan los tres casos comentados anteriormente.

2.a Neumann-Neumann:

$$\frac{1}{2}\begin{bmatrix} 1 & -1 & & \dots & & 0 \\ -1 & 2 & -1 & \dots & & 0 \\ \vdots & & & & & \vdots \\ 0 & & \dots & -1 & 2 & -1 \\ 0 & & \dots & & -1 & 1 \end{bmatrix} \begin{bmatrix} \sigma_1 \\ \sigma_1 \\ \vdots \\ \sigma_{N-1} \\ \sigma_N \end{bmatrix} = \begin{bmatrix} \tilde{\Delta}_1(\theta_n) \\ \tilde{\Delta}_2(\theta_n) \\ \vdots \\ \tilde{\Delta}_{N-1}(\theta_n) \\ \tilde{\Delta}_N(\theta_n) \end{bmatrix}$$

2.b Neumann-Dirichlet:

$$\frac{1}{2}\begin{bmatrix} 3 & -1 & & \dots & & 0 \\ -1 & 2 & -1 & \dots & & 0 \\ \vdots & & & & & \vdots \\ 0 & & \dots & -1 & 2 & -1 \\ 0 & & \dots & & -1 & 1 \end{bmatrix} \begin{bmatrix} \sigma_1 \\ \sigma_1 \\ \vdots \\ \sigma_{N-1} \\ \sigma_N \end{bmatrix} = \begin{bmatrix} \tilde{\Delta}_1(\theta_n) \\ \tilde{\Delta}_2(\theta_n) \\ \vdots \\ \tilde{\Delta}_{N-1}(\theta_n) \\ \tilde{\Delta}_N(\theta_n) \end{bmatrix}$$

2.c Dirichlet-Dirichlet:

$$\frac{1}{2}\begin{bmatrix} 3 & -1 & & \dots & & 0 \\ -1 & 2 & -1 & \dots & & 0 \\ \vdots & & & & & \vdots \\ 0 & & \dots & -1 & 2 & -1 \\ 0 & & \dots & & -1 & 3 \end{bmatrix} \begin{bmatrix} \sigma_1 \\ \sigma_1 \\ \vdots \\ \sigma_{N-1} \\ \sigma_N \end{bmatrix} = \begin{bmatrix} \tilde{\Delta}_1(\theta_n) \\ \tilde{\Delta}_2(\theta_n) \\ \vdots \\ \tilde{\Delta}_{N-1}(\theta_n) \\ \tilde{\Delta}_N(\theta_n) \end{bmatrix}$$

[4]

Se puede comprobar que las matrices de del vértice interior y Neumann-Neumann son singulares. Ambas tienen como espacio nulo el vector $l = [1,1,\dots,1]^T$, así que se añade la restricción de que el vector σ sea ortogonal al espacio nulo, o lo que es lo mismo, que la suma de todas sus componentes sea nula. De esta forma sí se obtiene una solución única y se puede obtener el valor del flujo, con el que calcular el error. [4]

Finalmente, aquí es donde se consiguen ver los puntos favorables de este método, ya que, en lugar de tener que lidiar con el problema completo para construir los flujos, se tienen unos sistemas definidos para cada nodo, que pueden llegar a tener matrices de coeficientes iguales, luego algunos de ellos se podrán llegar a resolver simultáneamente.[4]

4 Implementación en MATLAB

Una vez se ha descrito el Método del Residuo Equilibrado, se procede a trazar su introducción en MATLAB. En primer lugar, se enumerarán cada una de las entradas que se obtienen desde el programa de elementos finitos. Después, se verán cada una de las partes del código, acompañadas de los razonamientos y expresiones que se usan en ellas, conformando un código sobre el que se optimizará más tarde.

4.1 Datos de entrada al código de MATLAB

Para poder obtener una estimación del error, según este método, se reutilizan las siguientes matrices del programa de elementos finitos. Para poner expresar las dimensiones de las matrices se considera un mallado genérico de elementos triangulares compuesto por n_T elementos y n_N nodos.

4.1.1 Información del mallado

Estas matrices definen completamente el mallado y proceden del programa de la obtención de la discretización del dominio, que se realiza justo antes de obtener la solución aproximada.

- `T`, matriz de triangulación. Tiene una dimensión de $3 \times n_T$ y contiene los índices de los nodos que forman cada elemento. Siendo la columna k la que contiene los índices de los nodos del elemento k, $[n_{1,k}, n_{2,K}, n_{3,k}]^T$.

$$\texttt{T} = \begin{bmatrix} n_{1,1} & n_{1,2} & & n_{1,n_T} \\ n_{2,1} & n_{2,2} & \dots & n_{2,n_T} \\ n_{3,1} & n_{3,2} & & n_{3,n_T} \end{bmatrix}$$

- `Z`, matriz de coordenadas. Matriz de $2 \times n_N$ en la se incorporan las coordenadas de todos los nodos, teniendo en la columna n la coordenada x e y del nodo n.

$$\texttt{Z} = \begin{bmatrix} x_1 & x_2 & \dots & x_{n_N} \\ y_1 & y_2 & \dots & y_{n_N} \end{bmatrix}$$

- `B`, matriz de contorno. Matriz de $2 \times L_{front}$ con la que se establecen los lados que componen la frontera indicando sus dos extremos, estando contenidos los índices de los nodos de los extremos del lado j de la frontera columna j. Su dimensión depende

del número de lados que compongan la frontera, cuyo número será la suma de los lados de frontera Dirichlet y los de frontera Neumann, $L_{front} = L_N + L_D$.

$$\mathtt{B} = \begin{bmatrix} n_{l,1} & n_{l,2} & \dots & n_{l,n_{front}} \\ n_{r,1} & n_{r,2} & \dots & n_{r,n_{front}} \end{bmatrix}$$

- **IN**, índices de frontera Neumann. Vector de L_N componentes con los índices de los lados de la frontera que son de tipo Neumann.

$$\mathtt{IN} = [\gamma_{N,1}, \gamma_{N,2}, \dots, \gamma_{N,L_N}]$$

- **ID**, índices de frontera Dirichlet. Vector de L_D componentes con los índices de los lados de la frontera que son de tipo Dirichlet.

$$\mathtt{ID} = [\gamma_{D,1}, \gamma_{D,2}, \dots, \gamma_{D,L_D}]$$

- **ite**, matriz de vecindad. Tiene una dimensión de $n_T \times 3$ y se utiliza para cuál es el triángulo vecino a cada lado todos los elementos, o si es frontera, de la siguiente forma:

$$\mathtt{ite}(i,j) = \begin{cases} k > 0 & \text{, el lado } j \text{ del triángulo } i \text{ es común al triángulo } k \\ k < 0 & \text{, el lado } j \text{ del triángulo } i \text{ es el lado } -k \text{ de la frontera} \end{cases}$$

- **iteb**, matriz complementaria de vecindad. Al igual que *ite*su tamaño es de $n_T \times 3$ y ,como su nombre indica, añade información que no aporta la matriz de vecindad sobre cuál es el lado del triángulo vecino con el que se coincide.

$$\mathtt{iteb}(i,j) = \begin{cases} l > 0 & \text{, el lado } j \text{ del triángulo } i \text{ es el lado } l \text{ del triángulo } k \\ l = 0 & \text{, el lado } j \text{ del triángulo } i \text{ es frontera, luego no hay vecino} \end{cases}$$

4.1.2 Datos extraídos del cálculo de la solución aproximada

Estas matrices se calculan en el programa de resolución del problema de elementos finitos, y se pueden reutilizar para obtener una estimación del error. Reducen el tiempo de la obtención del mismo, ya que no hay que volver a calcularlas

- **u**, solución en los nodos. Vector de n_N componentes que contiene el valor de la solución aproximada en cada nodo.

$$\mathtt{u} = [u_1, u_2, \dots, u_{n_N}]^T$$

- **Ae**. Matriz de $3n_T \times e$ compuesta de n_T submatrices 3×3, una para cada elemento.

$$\mathtt{Ae} = \begin{bmatrix} B_1(\theta_i,\theta_j) \\ B_2(\theta_i,\theta_j) \\ \vdots \\ B_{n_T}(\theta_i,\theta_j) \end{bmatrix}$$

$$B_K(\theta_i,\theta_j) = \begin{bmatrix} B_K(\theta_1,\theta_1) & B_K(\theta_1,\theta_2) & B_K(\theta_1,\theta_3) \\ B_K(\theta_2,\theta_1) & B_K(\theta_2,\theta_2) & B_K(\theta_2,\theta_3) \\ B_K(\theta_3,\theta_1) & B_K(\theta_3,\theta_2) & B_K(\theta_3,\theta_3) \end{bmatrix}$$

- `fe`. Matriz de $3 \times n_T$ que contiene las integrales sobre cada elemento de $f\theta_n$ para todos los triángulos.

$$\texttt{fe} = \begin{bmatrix} (f,\theta_1)_1 & (f,\theta_1)_2 & (f,\theta_1)_{n_T} \\ (f,\theta_2)_1 & (f,\theta_2)_2 & (f,\theta_2)_{n_T} \\ (f,\theta_3)_1 & (f,\theta_3)_2 & (f,\theta_3)_{n_T} \end{bmatrix}$$

- `ge`. Matriz de $2 \times L_N$ que contiene la integral de $g\theta_n$ sobre todos los lados de frontera Neumann para sus dos extremos, siendo la columna j el valor de la integral para el lado de la frontera cuyo indice está en la posición j de `IN`.

$$\texttt{ge} = \begin{bmatrix} \int_{\gamma_{N_1}} g\theta_l\, ds & \int_{\gamma_{N_2}} g\theta_l\, ds & \dots & \int_{\gamma_{N_{LN}}} g\theta_l\, ds \\ \int_{\gamma_{N_1}} g\theta_r\, ds & \int_{\gamma_{N_2}} g\theta_r\, ds & \dots & \int_{\gamma_{N_{LN}}} g\theta_r\, ds \end{bmatrix}$$

  donde los subíndices r y l denotan al nodo inicial y final de cada lado, respectivamente.

4.2 Procedimiento inicial en MATLAB

Se describirán los pasos que se han seguido al programar en MATLAB con el fin de implementar el Método del Residuo Equilibrado. Este primer código, llamado versión 1.0, se elabora siguiendo la descripción del apartado anterior con la idea de poder entender cómo se introduce cada una de las variables del problema a MATALB, y el uso que se les da para obtener el resultado final. Todo ello sin tener en mente el tiempo que requieren para ser calculado, ya que, el código ideado aquí servirá como base, de estructura y de variables necesarias, para todas sus modificaciones posteriores.

1. Manipulación de los datos del mallado. Aquí se obtiene `NT` y `Nn`, número de triángulos y nodos, que serán necesarios para conocer las dimensiones de las matrices que se tendrán que crear más tarde. Estas constantes son fáciles de obtener mediante el uso del comando **size** sobre una de las filas de las matrices `T` y `z`.
Además se obtiene la longitud de cada lado de los triángulos y se guarda en `Ck`, una matriz de $3 \times n_T$, quedando en la columna j las longitudes de los lados en este orden $[L_1,L_2,L_3]^T$, tomando los lados de los triángulos como se ve en la Figura 4.1. Su cálculo es trivial, ya que, únicamente hay que calcular la longitud de un segmento.

$$C_K^{\gamma} = \sqrt{\left(x_f - x_i\right)^2 + \left(y_f - y_i\right)^2}$$

siendo (x_i, y_i) y (x_f, y_f), las coordenadas los extremos del lado γ del triángulo K.

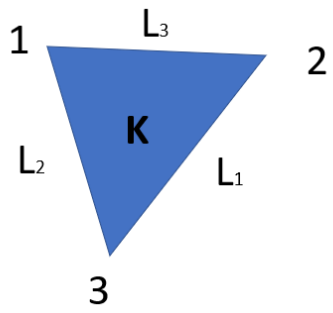

Figura 4.1 Triángulo K genérico.

Código 4.1 Datos del Mallado.

```
NT=length(T(1,:)); %Numero de triangulos
Nn=length(z(1,:)); %Numero de nodos

Ck=zeros(3,NT);%Longitud de los lados. Ck(L,k) lado L del triangulo k
for k=1:NT %Para cada triangulo
    indices=T(:,k); %Indices de los nodos del triangulo k
    %- - Longitud de lados - -
    Ck(:,k)=[sqrt((z(1,indices(3))-z(1,indices(2)))^2...
       ...+(z(2,indices(3))-z(2,indices(2)))^2);
          sqrt((z(1,indices(1))-z(1,indices(3)))^2...
       ...+(z(2,indices(1))-z(2,indices(3)))^2);
             sqrt((z(1,indices(2))-z(1,indices(1)))^2...
       ...+(z(2,indices(2))-z(2,indices(1)))^2)];
end
```

2. Cálculo del gradiente de u_x. Se guardará en la matriz `gradux` de $2 \times n_T$, con una columna para cada triángulo, siendo la componente de la primera fila la componente según X y la otra fila la de Y. En primer lugar, se verá la expresión para obtener el valor del gradiente de la solución aproximada para un triángulo genérico. Para ello, se utiliza el siguiente cambio de variables para transformar todos los triángulos en uno rectángulo con vértices en (0,0), (1,0) y (1,1) en el nuevo sistemas de coordenadas.

$$\begin{bmatrix} x(\xi,\eta) \\ y(\xi,\eta) \end{bmatrix} = \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} + \begin{bmatrix} x_2 - x_1 & x_3 - x_1 \\ y_2 - y_1 & y_3 - y_1 \end{bmatrix} \begin{bmatrix} \xi \\ \eta \end{bmatrix}$$

La expresión de $u_x(\xi,\eta)$ sería

$$u_x(\xi,\eta) = [u_1, u_2, u_3] \begin{bmatrix} \varphi_1(\xi,\eta) \\ \varphi_2(\xi,\eta) \\ \varphi_3(\xi,\eta) \end{bmatrix}$$

siendo φ_n las funciones base o de forma de cada nodo en el triángulo de vértices (0,0), (1,0) y (1,1).

$$N(\xi,\eta) = \begin{bmatrix} \varphi_1(\xi,\eta) \\ \varphi_2(\xi,\eta) \\ \varphi_3(\xi,\eta) \end{bmatrix} = \begin{bmatrix} 1-\xi-\eta \\ \xi \\ \eta \end{bmatrix}$$

Así que aplicando la regla de la cadena el gradiente sería de la siguiente forma:

$$\begin{bmatrix} \dfrac{\partial u_x}{\partial x} \\ \dfrac{\partial u_x}{\partial y} \end{bmatrix} = \begin{bmatrix} \dfrac{\partial u_x}{\partial \xi}\dfrac{\partial \xi}{\partial x} + \dfrac{\partial u_x}{\partial \eta}\dfrac{\partial \eta}{\partial x} \\ \dfrac{\partial u_x}{\partial \xi}\dfrac{\partial \xi}{\partial y} + \dfrac{\partial u_x}{\partial \eta}\dfrac{\partial \eta}{\partial y} \end{bmatrix} = [u_1,u_2,u_3]\left[N_\xi \xi_x + N_\eta \eta_x\,, N_\xi \xi_y + N_\eta \eta_y\right]$$

siendo

$$N_\xi = [-1,1,0]^T \quad , \quad N_\eta = [-1,0,1]^T$$

$$\xi_x = \frac{y_3 - y_1}{J} \qquad \eta_x = -\frac{y_2 - y_1}{J}$$

$$\xi_y = -\frac{x_3 - x_1}{J} \qquad \eta_y = \frac{x_2 - x_1}{J}$$

$$J = (x_2 - x_1)(y_3 - y_1) - (x_3 - x_1)(y_2 - y_1)$$

En definitiva, se tiene una expresión universal gracias a que el gradiente de esta forma. Así que se realizará un bucle con el que se obtendrán sus dos componentes para cada triángulo. Cabe destacar que el valor del gradiente es constante en cada elemento gracias a que las funciones de forma con las que se trabaja en este caso son lineales.

Código 4.2 Gradiente de u_x.

```
gradux=zeros(2,NT);%Fila 1 u_x - - -Fila 2 u_y
for k=1:NT %Para cada triangulo
    indices=T(:,k); %Indices de los nodos del triangulo k
    Nxi=[-1;1;0];
    Neta=[-1;0;1];

    denominador=(z(1,indices(2))-z(1,indices(1)))*(z(2,indices(3))-...
       ...z(2,indices(1)))-(z(1,indices(3))-z(1,indices(1)))*...
       ...(z(2,indices(2))- z(2,indices(1)));

    xix=(z(2,indices(3))-z(2,indices(1)))/denominador;
    etax=-1*(z(2,indices(2))-z(2,indices(1)))/denominador;
    xiy=-1*(z(1,indices(3))-z(1,indices(1)))/denominador;
    etay=(z(1,indices(2))-z(1,indices(1)))/denominador;

    gradux(1,k)= [u(indices(1)) u(indices(2)) u(indices(3))]*...
       ...(Nxi*xix+Neta*etax);
    gradux(2,k)= [u(indices(1)) u(indices(2)) u(indices(3))]*...
       ...(Nxi*xiy+Neta*etay);

end
```

3. Obtención del momento del flujo aproximado, $\tilde{\mu}^{\gamma}_{K,n}$. Este valor es necesario para obtener el valor de los momentos del flujo, además se podrá utilizar para calcular otras variables, como se verá más tarde. Por su defición habrá que calcular uno para cada lado y cada función base de cada nodo, luego se almacenará en `muAprox`, una matriz $(3 \times 3n_T\ )$. Teniendo una submatriz (3×3) referida a cada triángulo, donde cada elemento (i,j) tendrá el momento aproximado del lado j de la función base i.

$$\tilde{\mu}^{\gamma}_{K,n} = \int_{\gamma} \theta_n n_K \cdot \nabla u_x \, ds$$

En esta integral, el producto escalar entre el gradiente y la normal exterior será constante, al ser rectas los lados de los elementos y el gradiente constante en cada elemento. Consecuentemente lo único que cambia de valor a lo largo de cada lado serán las funciones base, así que se integrarán y posteriormente se multiplicarán por el valor del producto escalar citado anteriormente. Al igual que se ha hecho con la expresión del gradiente, se recurre al cambio de variables a (η,ξ) para obtener una expresión universal para todos los triángulos. Al integrar las funciones bases del caso de (η,ξ) se tienen los siguientes resultados.

$$\int_{L_j} \varphi_i \, ds = \begin{bmatrix} 0 & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} & 0 \end{bmatrix}$$

Para hacer el cambio de variables sólo habrá que multiplicar cada columna por la longitud de su respectivo lado, que matricialmente se expresa como el producto por la derecha con una matriz diagonal cuyas componentes son las longitudes de los lados.

$$\int_{L_j} \theta_i \, ds = \begin{bmatrix} 0 & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} & 0 \end{bmatrix} \begin{bmatrix} C_1 & 0 & 0 \\ 0 & C_2 & 0 \\ 0 & 0 & C_3 \end{bmatrix}$$

donde C_l son las longitudes de los lados.
La normal exterior de cada lado se calcula utilizando expresiones básicas de la geometría. Para ello será necesario definir los nodos iniciales y finales de cada lado, en los vectores `xi` y `xf`. Con ellos, se almacenan las dos componentes de la normal en `nlado`. Por último se obtiene el valor de la integral como el producto de la integral de la función base con el producto escalar, calculado como la multiplicación de un vector fila con un vector columna.

Código 4.3 Cálculo del momento del flujo aproximado.

```
IntThetan=[0 0.5 0.5; 0.5 0 0.5; 0.5 0.5 0 ]'; %En eta xi

muAprox=zeros(3,3*NT);
for k=1:NT

    indices=T(:,k);

    xi=[z(1,indices(2)) z(1,indices(3)) z(1,indices(1))]; %x inicial
```

```
    xf=[z(1,indices(3)) z(1,indices(1)) z(1,indices(2))]; %x final
    yi=[z(2,indices(2)) z(2,indices(3)) z(2,indices(1))]; %y inicial
    yf=[z(2,indices(3)) z(2,indices(1)) z(2,indices(2))]; %y final

    for L=1:3
        nlado=[yf(L)-yi(L) , -1*(xf(L)-xi(L))]/sqrt((yf(L)-yi(L))^2+...

           ...(xf(L)-xi(L))^2); %normal del lado L
        muAprox(:,3*(k-1)+L)=(nlado*gradux(:,k))*...
           ...IntThetan(:,L)*Ck(L,k);
    end
end
```

4. Cálculo de $\tilde{\Delta}_{K,n}$. Cada nodo de cada triángulo tiene un término independiente de los sistemas de ecuaciones para obtener $\sigma_{K,n}$. Así que se obtienen todo ellos y se almacenan en la matriz `deltagorro` , que tiene una estructura igual a la de `T`, pero en lugar de almacenar el índice de los nodos, son los $\tilde{\Delta}_{K,n}$. Se recuerda la expresión de $\tilde{\Delta}_{K,n}$:

$$\tilde{\Delta}_K(\theta_n) = B_K(u_x,\theta_n) - (f,\theta_n)_K - \int_{\partial K} \left\langle \frac{\partial u_x}{\partial n_K} \right\rangle \theta_n \, ds$$

siendo

$$\left\langle \frac{\partial u_x}{\partial n_K} \right\rangle = \begin{cases} \frac{1}{2} n_K \cdot \{(\nabla u_x)_K + (\nabla u_x)_{K'}\} & \text{sobre} \quad \partial K \cap \partial K' \\ n_K \cdot (\nabla u_x)_K & \text{sobre} \quad \partial K \cap \Gamma_D \\ g & \text{sobre} \quad \partial K \cap \Gamma_N \end{cases}$$

El término de $B_K(u_x,\theta_n)$ se obtiene de multiplicar la submatriz $\texttt{Ae}_K$ con las componentes de `u` en los nodos del triángulo K, mientras que el de $(f,\theta_n)_K$ se obtiene directamente de la columna K de la matriz `fe`. Por último, el otro término se puede calcular aprovechando la matriz del momento de flujo aproximado `muAprox`, calculada anteriormente, y la matriz de `ge`. Se calculan lado a lado y se van sumando en la columna K de la matriz `Intuxnk`, en la que se almacena el valor de $\int_{\partial K} \left\langle \frac{\partial u_x}{\partial n_K} \right\rangle \theta_n \, ds$. Según el tipo de lado que sea se pueden tener las siguientes situaciones:

- Lado interior. Se tiene un triángulo vecino, del que se sabe cuál es y el lado que es en éste gracias a las matrices `ite` y `iteb` . El valor a calcular es la integral sobre el lado del producto escalar entre la normal al lado por la media del gradiente en los dos triángulos en los que está el lado. En sí, la parte correspondiente al triángulo K se tiene calculado en la columna correspondiente de `muAprox`, y para obtener la contribución del triángulo vecino sólo habrá que realizar unos cambios a la columna de `muAprox` correspondiente al mismo lado pero en el triángulo vecino K'. La integral sobre el lado de la función base, como se ha visto antes, únicamente depende de su longitud, que obviamente es la misma en ambos triángulos. Luego la única diferencia que se tiene es que $\tilde{\mu}^{\gamma}_{K',n}$ está calculado con $n_{K'}$, al ser el mismo lado, n_K y n_K serán iguales pero de signo opuesto.

$$\int_{\gamma=\partial K\cap\partial K'} \frac{1}{2} n_K \cdot (\nabla u_x)_K \, \theta_n \, ds + \int_{\gamma=\partial K\cap\partial K'} \frac{1}{2} n_K \cdot (\nabla u_x)_{K'} \, \theta_n \, ds = \frac{1}{2} \left(\tilde{\mu}^{\gamma}_{K,n} - \tilde{\mu}^{\gamma}_{K',n} \right)$$

Para que tenga sentido expresar el término de esa forma es necesario introducir la otra modificación. Con esta se consigue que cada una de las componentes de las columnas `muAprox` referidas a ese lado se refieran al mismo nodo en cada una de sus componentes. Realmente, uno de los nodos de cada vector no coincide con el otro, concretamente el que está en el vértice opuesto al lado en cuestión. Pero como el valor asociado a éstos en `muAprox` es nulo, porque la integral de la función base es 0, al uso son iguales. Para entender mejor este cambio se ejemplifica un caso en la Figura 4.2.

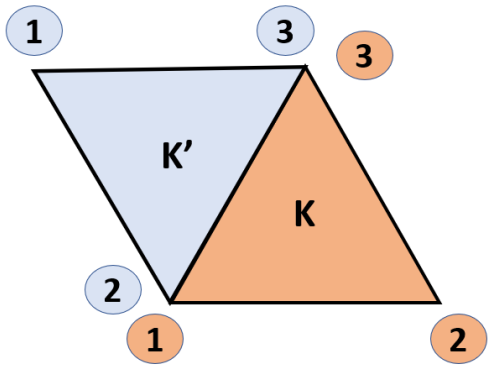

Figura 4.2 Caso en el que el lado 2 de K coincide con el lado 1 de su vecino.

En este caso se tendría los siguientes vectores de momentos:

$$\tilde{\mu}^{L_2}_{K,n} = \begin{bmatrix} \tilde{\mu}^{L_2}_{K,1} \\ 0 \\ \tilde{\mu}^{L_2}_{K,3} \end{bmatrix} \qquad \tilde{\mu}^{L_1}_{K',n} = \begin{bmatrix} 0 \\ \tilde{\mu}^{L_1}_{K',2} \\ \tilde{\mu}^{L_1}_{K',3} \end{bmatrix}$$

En este caso, para que para los valores de los mismo nodos coincidan, se debe intercambiar la primera componente con la primera, en el vector del triángulo vecino.
Si se realiza este proceso para los 9 casos posibles se llega a construir una llamada `Ordenacion` matriz de (3×9), que se divide en 3 submatrices, una para cada lado del triángulo K, en la que cada columna j está referida al orden que deben tener las componentes del vector con los momentos aproximados del triangulo vecino en el lado j.

$$\texttt{Ordenacion} = \left[\begin{array}{ccc|ccc|ccc} 1 & 2 & 3 & 2 & 3 & 1 & 3 & 1 & 2 \\ 3 & 1 & 2 & 1 & 2 & 3 & 2 & 3 & 1 \\ 2 & 3 & 1 & 3 & 1 & 2 & 1 & 2 & 3 \end{array}\right]$$

Con el uso de la columna adecuada esta matriz para reordenar las componentes de $\tilde{\mu}^{\gamma}_{K',n}$ sí se puede decir que se pueden sumar los vectores citados anteriormente para obtener el valor de $\int_{\partial K} \left\langle \frac{\partial u_x}{\partial n_K} \right\rangle \theta_n \, ds$ como una combinación lineal de dos columnas de `muAprox`.

- Lado con frontera Dirichlet. La integral sobre este lado se obtiene de la misma forma que el caso anterior, sólo tomando el triángulo K. Es decir, es igual a la

columna correspondiente a ese lado en `muAprox`.

$$\int_{\gamma=\partial K\cap\Gamma_D} n_K\cdot(\nabla u_x)_K\,\theta_n\,ds=\tilde{\mu}^{\gamma}_{K,n}$$

- Lado con frontera Neumann. El valor de la integral del lado se obtiene de la matriz `ge`. Esta matriz únicamente tiene dos componentes para cada lado, debido a que la otra es nula, luego el orden en el que se deben de colocar depende del lado del triángulo K que se trate. Dándose las tres situaciones siguientes:
  -Lado 1 de K, entonces la componente asociada al nodo 1 será nula.
  -Lado 2 de K, la segunda componente será cero.
  -Lado 3 de K, la componente del tercer nodo será nula.

Otro aspecto importante que falta por ser tratado es la forma en la que se identifica el tipo de lado que se tiene. El primer paso es comprobar si es frontera con el signo de la componente de `ite` asociada al lado en cuestión, si es positiva se trata de un lado interior. En caso contrario, contiene el opuesto al lado de la frontera que es, por lo que se busca ese mismo valor en `IN` mediante la orden **find** de MATLAB. Si el vector resultante de esa instrucción está vacío, el lado es de tipo Dirichlet. Mientras que si no está vacío es el único caso restante, lado Neumann, en éste habrá que buscar la posición que ocupa ese lado de la frontera en el vector `IN`, que determina la columna de `ge` que le corresponde.

Código 4.4 Términos independientes de los sistemas.

```
Bk=zeros(3,NT);
fek=Bk;
for k=1:NT
    indices=T(:,k);
    Bk(:,k)=Ae([3*k-2,3*k-1,3*k],:)*[u(indices(1));...
       ... u(indices(2)); u(indices(3))];
    fek(:,k)=fe(:,k);
end

Intuxnk=zeros(3,NT);

          %L1 de k   %L2 de k  %L3 de k
Ordenacion=[ 1 2 3    2 3 1     3 1 2; %theta 2 de k
             3 1 2     1 2 3     2 3 1; %theta 2 de k
             2 3 1     3 1 2     1 2 3];%theta 3 de k
          %L1,L2,L3 de k'
for k=1:NT
    IntEnk=zeros(3,1); %vamos sumando cada lado
    for L=1:3

        %es frontera?
        if ite(k,L)<0 %es frontera, lado -ite(j,L) del contorno
            %tipo de frontera
            if isempty(find(IN==-ite(k,L)))
                %es frontera Dir
                IntEnk=IntEnk+muAprox(:,3*(k-1)+L);
            else % es fronetra Neu
```

```
                LNeu=find(IN==-ite(k,L));
                if L==1
                    IntEnk=IntEnk+[0;ge(:,LNeu)];
                elseif L==2
                    IntEnk=IntEnk+[ge(2,LNeu);0;ge(1,LNeu)];
                else
                    IntEnk=IntEnk+[ge(:,LNeu);0];
                end
            end
        else %no es frontera
            Tvec=ite(k,L);%Triangulo vecino por el lado L
            Lvec=iteb(k,L);%Lado del triangulo vecino
            muAproxvec=muAprox(Ordenacion(:,3*(L-1)+Lvec),...
               ...3*(Tvec-1)+Lvec);
            IntEnk=IntEnk+0.5*(-muAproxvec+muAprox(:,3*(k-1)+L));
        end
    Intuxnk(:,k)=IntEnk;
    end
end

deltagorro=Bk-fek-Intuxnk;
```

5. Obtención de los multiplicadores de Lagrange $\sigma_{K,n}$. Al igual que en el caso de $\tilde{\Delta}_{K,n}$, se almacenan de forma similar a la matriz `T`, en una matriz de $(3 \times n_T)$ con una columna para cada elemento y una fila para cada uno de los nodos. Aunque durante su calculo se almacena en una matriz auxiliar de distinto tamaño, que se verá más tarde.
Una forma intuitiva de resolver los sistemas que surgen es ir nodo por nodo resolviendo uno a uno todos los sistemas que surgen. Para cada nodo, mediante el comando **find** se busca sobre la matriz `T` el índice del nodo considerado, obteniéndose dos vectores, uno con las columnas en las que se encuentra, es decir, los elementos en los que está; y otro con las filas en las que se encuentra, o lo que es lo mismo, el número del nodo que ocupa en cada triángulo. La longitud de estos vectores será el número de elementos en los que está el nodo en cuestión. Además de eso, ya se puede crear el vector de términos independientes, asignando a cada una de sus componentes el valor de $\tilde{\Delta}_{K,n}$ referente al triángulo y nodo que indica la componente de la misma posición de los vectores `filas` y `columnas`.
Sabiendo el término independiente, para resolver el sistema de ecuaciones sólo queda conocer la matriz de coeficientes, que está definida por el tipo de nodo que es y el tamaño del término independiente, o lo que es lo mismo, el número de elementos en los que está ese nodo. Conocer qué tipo de nodo es se realiza buscando con el comando **find** en la matriz de contorno, `B`, el índice del nodo en cuestión. Si devuelve dos vectores vacíos, el nodo es interior. En el caso contrario es frontera, por lo que hay que volver a buscar en `ID` los índices de los lados de la frontera en los que se encuentra, que será el vector que contiene las columnas al usar **find** en `B`. Si devuelve un vector vacío, el nodo es Nuemann-Neumann, si tiene una componente es Nuemann-Dirichlet y si tiene dos es Dirichlet-Dirichlet. Éste razonamiento se debe a que sabiendo que es frontera, sólo de sus lados pueden ser frontera y sólo hay dos tipos de frontera, luego si no se encuentran en la matriz de lados Dirichlet, estarán en la de lados Neumann.
Con el tipo y el término independiente, se resuelve el sistema mediante unas funciones

auxiliares. En el que se tiene un vector, con la longitud, $NTri_n$, del número de triángulos en los que está cada nodo. Estas funciones generan la matriz de coeficientes asociada a ese tipo de nodo y de las dimensiones adecuadas. Con respecto a la resolución de cada uno de los sistemas, en el caso de nodo interior y Neumann-Neumann se utiliza la factorización LU, mientras que en los otros dos se resuelve con la factorización de Cholesky al ser las matrices simétricas y definidas positivas.
Para poder resolver así todos los sistemas asociados a cada nodo, el vector que dan las funciones se almacenan en la columna n de matriz auxiliar `sigmaden`, de dimensión $(NTri_{max} \times n_N)$. Del mismo modo se almacenan los triángulos a los que están referidos cada multiplicador, contenidos en `columnas`, en la matriz `Tsigma`, siendo `sigmaden`(i,j) el valor del multiplicador de Lagrange del nodo de índice j en el triángulo `Tsigma`(i,j).
Para calcular el valor máximo de $NTri_n$ se busca el máximo sobre un vector columna creado con el comando **sparse** con el que se obtiene el número de elementos en los que está cada nodo. A este comando hay que darle como entrada un vector o matriz de filas, otro de columnas y otro de valores. Si se introducen varios veces valores en una misma componente de la matriz dispersa, éstos se van sumando. Luego como entradas a **sparse** se da la matriz de triangulación ,`T` , para determinar las filas, un vector unidad para las columnas y un vector unidad para los valores, de forma que cada vez que aparece el índice de un nodo en `T` se suma 1 en esa fila del vector que se obtiene.
Finalmente, cuando ya se hayan resuelto los sistemas de ecuaciones de todos los nodos, sólo queda asignar los multiplicadores de `sigmaden` a una matriz $(3 \times n_T)$, llamada `sigma`. Se hace elemento a elemento, dentro de cada uno se va nodo a nodo, se **find** sobre `Tsigma` en la fila del nodo en cuestión para saber qué elemento va en la posición de ese elemento y nodo.

Código 4.5 Obtención de los multiplicadores de Lagrange.

```
nmax=max(sparse(T,ones(3*NT,1),ones(3*NT,1)));
sigmaden=zeros(nmax,Nn);
Tsigma=sigmaden;

for n=1:Nn
    [filas,columnas]=find(T==n);
        %columnas son los triangulos en los que esta n
        %filas son el nodo que son de cada triangulo
    N=length(columnas); %numero de triangulos en los que esta
    Tsigma(1:N,n)=columnas;
    deltagorron=zeros(N,1);

    for j=1:N
        deltagorron(j)=deltagorro(filas(j),columnas(j));
    end

    %tipo de nodo
    [~,frontera]=f ind(B==n);
    if isempty(frontera)
        %no es frontera,
        sigmaden(1:N,n)=VerticeInterior(deltagorron);
```

```
    else %es frontera,hay que ver el tipo
        [~,frontera]=find(ID==frontera);
        if isempty(frontera) %tipo N-N
            sigmaden(1:N,n)=VerticeNeuNeu(deltagorron);
        elseif length(frontera)==1 %tipo N-D
            sigmaden(1:N,n)=VerticeDirNeu(deltagorron);
        else %tipo D-D
            sigmaden(1:N,n)=VerticeDirDir(deltagorron);
        end
    end
end

%- - - -
%se asigna a sigma
sigma=zeros(3,NT);
for k=1:NT %cada triangulo
    indices=T(:,k);
    for n=1:3 %en cada nodo
        fila=find(Tsigma(:,indices(n))==k);
        sigma(n,k)=sigmaden(fila,indices(n));
    end
end


function [ sigma ] = VerticeNeuNeu( TIndep )
%VERTICE DE FRONTERA NEUMANN-NEUMANN,singular entonces (N+1,1) ec
    extra del espacio nulo
N=length(TIndep);
if N==1
    sigma=TIndep/1*2;
else
    b=[TIndep; 0];
    d0=[1 2*ones(1,N-2) 1];
    d1=-1*ones(1,N-1);
    M=zeros(N+1,N);
    M(1:N,1:N)=diag(d0,0)+diag(d1,-1)+diag(d1,1);
    M(N+1,:)=ones(1,N);
    M=M/2;
    [L,U,P] = lu(M); %Ax=b PA=LU LUx=Pb Ly=Pb Ux=y
    y=L\(P*b);
    sigma=U\y;

end
end

function [sigma]=VerticeInterior(TIndep)
%VERTICE INTERIOR
%singular entonces (N+1,1) ec extra del espacio nulo
N=length(TIndep);
```

```
b=[TIndep; 0];
d0=2*ones(1,N);
d1=-1*ones(1,N-1);
Mint=zeros(N+1,N);
Mint(1:N,1:N)=diag(d0,0)+diag(d1,-1)+diag(d1,1);
Mint(1,N)=-1;
Mint(N,1)=-1;
Mint(N+1,:)=ones(1,N);
Mint=Mint/2;
[L,U,P] = lu(Mint); %Ax=b PA=LU LUx=Pb Ly=Pb Ux=y
y=L\(P*b);
sigma=U\y;

end

function [ sigma ] = VerticeDirDir( TIndep )
%VERTICE DE FRONTERA DIRICHLET-DIRICHLET
%MATRIZ SIMETRICA DEFINIDA POSITIVA => CHOLESKY

N=length(TIndep);
if N==1
    sigma=TIndep/3*2;
else
    d0=[3 2*ones(1,N-2) 3];
    d1=-1*ones(1,N-1);
    M=diag(d0,0)+diag(d1,-1)+diag(d1,1);
    M=M/2;
    R=chol(M); %A=R'R R'y=b Rx=y
    y=R'\TIndep;
    sigma=R\y;


end
end
function [ sigma ] = VerticeDirNeu( TIndep )
%VERTICE DE FRONTERA DIRICHLET-NEUMANN
%MATRIZ SIMETRICA DEFINIDA POSITIVA => CHOLESKY
N=length(TIndep);
if N==1
    sigma=TIndep/3*2;
else
    d0=[3 2*ones(1,N-2) 1];
    d1=-1*ones(1,N-1);
    M=diag(d0,0)+diag(d1,-1)+diag(d1,1);
    M=M/2;
    R=chol(M); %A=R'R R'y=b Rx=y
    y=R'\TIndep;
    sigma=R\y;

end
```

```
end
```

6. Construcción de los momentos de los flujos. Una vez se han calculado los valores de los multiplicadores y los momentos aproximados se puede tener el valor de $\mu^{\gamma}_{K,n}$, que será almacenado en la matriz `mu` de igual forma que se hace con `muAprox`, es decir, en una matriz $(3 \times 3n_T)$ compuesta de n_T submatrices de (3×3). Para explicar el procedimiento seguido se recuerda la expresión para obtener $\mu^{\gamma}_{K,n}$.

$$\mu^{\gamma}_{K,n} = \begin{cases} \frac{1}{2}\left(\sigma_{K,n} - \sigma_{K',n} + \tilde{\mu}^{\gamma}_{K,n} - \tilde{\mu}^{\gamma}_{K',n}\right) & \gamma = \partial K \cap \partial K \\ \int_{\gamma} g\theta_n \, ds & \gamma = \partial K \cap \Gamma_N \\ \sigma_{K,n} + \tilde{\mu}^{\gamma}_{K,n} & \gamma = \partial K \cap \Gamma_D \end{cases}$$

Al depender del lado y del elemento, se irá de lado a lado por cada uno de los elementos, o lo que es lo mismo, llenando una a una las columnas de la matriz que guarda el valor de los momentos del flujo. Se identificarán los lados de igual forma que en el caso del cálculo de $\tilde{\Delta}_{K,n}$ y el triángulo vecino y su lado mediante `ite` e `iteb`. Debe de destacarse que en el caso de lado interior, las componentes de $\tilde{\mu}^{\gamma}_{K',n}$ y las de $\sigma_{K',n}$ que se toman deben estar referidas a los nodos del triángulo K, es decir, se han de ordenar mediante la matriz `Ordenacion`, al igual que se hizo en el cálculo de los momentos del flujo. Mientras que cuando el lado sea frontera Neumann, se sigue el mismo procedimiento que en `muAprox`.

Código 4.6 Valor del flujo.

```
mu=zeros(3,3*NT);
for k=1:NT
    for L=1:3
        %ver tipo
        if ite(k,L)>0 %no es frontera
            Tvec=ite(k,L);%triangulo vecino
            Lvec=iteb(k,L);%lado del triangulo vecino
            sigmavec=sigma(Ordenacion(:,3*(L-1)+Lvec),Tvec);
            muAproxvec=muAprox(Ordenacion(:,3*(L-1)+Lvec),...
               ...3*(Tvec-1)+Lvec);
            mu(:,3*(k-1)+L)=0.5*(sigma(:,k)-sigmavec+...
               ..muAprox(:,3*(k-1)+L)-muAproxvec);
        elseif isempty(find(IN==-ite(k,L)))%es frontera, ver tipo
            %es frontera Dir
            mu(:,3*(k-1)+L)=sigma(:,k)+muAprox(:,3*(k-1)+L);
        else%es frontera Neu es dato
            LNeu=find(IN==-ite(k,L));
            if L==1
                mu(:,3*(k-1)+L)=[0;ge(:,LNeu)];
            elseif L==2
                mu(:,3*(k-1)+L)=[ge(2,LNeu);0;ge(1,LNeu)];
            else
                mu(:,3*(k-1)+L)=[ge(:,LNeu);0];
            end
```

```
        end
    end
end
```

7. Integral sobre los lados y estimación del error. Con todos los pasos anteriores ya se tienen todos los datos necesarios para definir la función del flujo sobre cada uno de los lados de los elementos, definida en la sección anterior como:

$$g_K|_\gamma = \frac{2}{h}\left\{\left(2\mu_{K,l}^\gamma - \mu_{K,r}^\gamma\right)\theta_l + \left(-\mu_{K,l}^\gamma + 2\mu_{K,r}^\gamma\right)\theta_r\right\}$$

El valor que interesa calcular sobre el flujo es su integral sobre los lados de todos los elementos, y teniendo en cuenta que es una función lineal, conociendo el valor del flujo en dos puntos y la longitud del lado se puede obtener el valor de la integral como la media entre los dos puntos multiplicado por la longitud del lado.
Los flujos en los extremos se guardarán en la matriz `gk` de $(2 \times 3n_T)$ con una columna para cada lado de cada elemento y una fila para el extremo inicial del lado y otra para el final. Se va llenando yendo de triángulo en triángulo y de lado en lado, definiendo g_K en los extremos del lado de cada iteración, es decir, uno para cuando $\theta_l = 0$ y el otro cuando $\theta_l = 0$.
En cuanto la integral se almacena en `Intgk`, un vector de n_T componentes, cada una de ellas asociada a un elemento. Para obtener su valor se realiza con un bucle para barrer todos los triángulos y se va sumando la aportación de cada lado como la media del flujo de los extremos por la longitud de los lados.

Código 4.7 Flujo y su integral sobre los lados.

```
gk=zeros(2,3*NT); %fila 1 valor en nodo inical, fila 2 valor en nodo
    final
for k=1:NT
    %lado 1   de 2 a 3
    %lado 2   de 3 a 1
    %lado 3   de 1 a 2
    ThetaInicial=[2,3,1];
    ThetaFinal=[3,1,2];
    for L=1:3
        gk(:,3*(k-1)+L)=2/Ck(L,k)*[2*mu(ThetaInicial(L),3*(k-1)+L)...
           ...-mu(ThetaFinal(L),3*(k-1)+L); -mu(ThetaInicial(L),...
           ...3*(k-1)+L)+2*mu(ThetaFinal(L),3*(k-1)+L)];
    end
end
%- - - - -
%integral de gk en el contorno de todos los triangulos
Intgk=zeros(1,NT);
for k=1:NT
    for L=1:3
        Intgk(k)=Intgk(k)+0.5*(gk(1,3*(k-1)+L)+...
           ...gk(2,3*(k-1)+L))*Ck(L,k);
    end
end
```

5 Optimización del rendimiento

5.1 Rendimiento de la versión 1.0

En la sección anterior se ha planteado un primer código que implementa el Método del Residuo Equilibrado, en esta se estudiará los tiempos de cálculos y se intentarán reducir todo lo posible.

En el desarrollo de esta sección se dividirán los tiempos de programa de MATLAB en los siguientes grupos:

- Parte 1. Cálculo de $\tilde{\Delta}_{K,n}$. Incluye desde el paso 1 hasta el 4 de la sección anterior.
- Parte 2. Obtención de $\sigma_{K,n}$, es en la que se resuelven todos los sistemas de ecuaciones que surgen en cada nodo para calculas los multiplicadores de Langrange.
- Parte 3. Cálculo de la integral sobre el contorno de cada elemento del flujo. En ella se obtiene el valor de los momentos, con los que se reconstruye el flujo en cada lado.

Estos tiempos se comprobarán para diferentes mallados, que se definirán por N, el número de divisiones de cada lado del rectángulo inicial, y la forma en la que se crea. Una de ellas será con un mallado en el que todos los elementos son triángulos rectángulos del mismo tamaño. Mientras que la otra será creada de forma aleatoria.

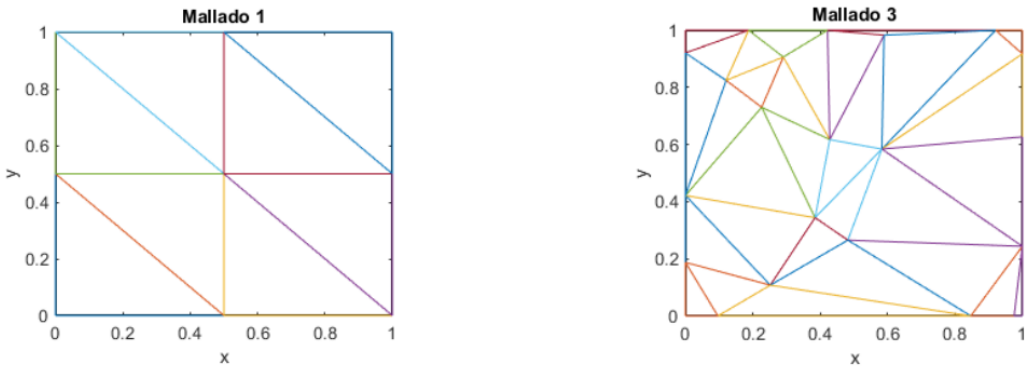

Figura 5.1 Ejemplos de mallados.

Como primera referencia de tiempos se tienen los de cálculo de la solución de elementos finitos para varios valores de N, que quedan recogidos a continuación en las Tablas 5.1 y 5.2, en las que se incluye también otros datos sobre la discretización del dominio en cuestión, como el número de elementos y de nodos.

En estas tablas, se observa que los tiempos del primer caso son más pequeños que los del segundo, esta diferencia se acentúa al crecer el refinamiento de la malla. Además, atendiendo al número de elementos de las mallas se ve que han tomado desde unas relativamente bastas, hasta unas con una cantidad de elementos varios órdenes de magnitud superior.

Tabla 5.1 Tiempos de cálculo de la solución de elementos finitos para red regular.

N	20	80	160	240	330	450
nº de triángulos	800	12800	51200	115200	217800	405000
nº de nodos	441	6561	25921	58081	109561	203401
t_{sol} (s)	0.4412	0.1751	0.8464	2.2335	4.2326	8.0890

Tabla 5.2 Tiempos de cálculo de la solución de elementos finitos para red generada aleatoriamente.

N	20	80	160	240	330	450
nº de triángulos	800	12800	51200	115200	217800	405000
nº de nodos	441	6561	25921	58081	109561	203401
t_{sol} (s)	1.2973	0.6582	3.1478	8.1834	18.6325	27.9692

Con todo esto ya si se puede empezar a comparar tiempo con la primera versión del código que se tiene. Teniendo en cuenta que se reutilizan datos del problema de elementos finitos y el mallado es ya conocido, se espera tener unos tiempos mucho más pequeños que los de referencia. Entonces, se utiliza el código midiendo los tiempos de cada parte para poder hacer una comparación.

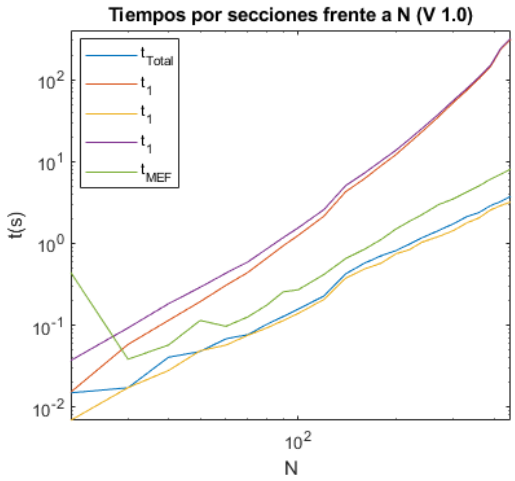

Figura 5.2 Tiempos de cálculo con mallados regulares en V 1.0.

Tabla 5.3 Tiempos de cálculo por partes para red regular en V 1.0 .

N	20	80	160	240	330	450
Parte 1(s)	0.0149	0.1020	0.5721	1.1652	2.1159	3.7622
Parte 2(s)	0.0152	0.6573	6.1568	22.9602	73.8507	311.7745
Parte 3(s)	0.0069	0.0923	0.4862	1.0292	1.7764	3.1908
Total(s)	0.0370	0.8516	7.2151	25.1546	77.7431	318.7275
Solución MEF(s)	0.4412	0.1751	0.8464	2.2335	4.2326	8.0890

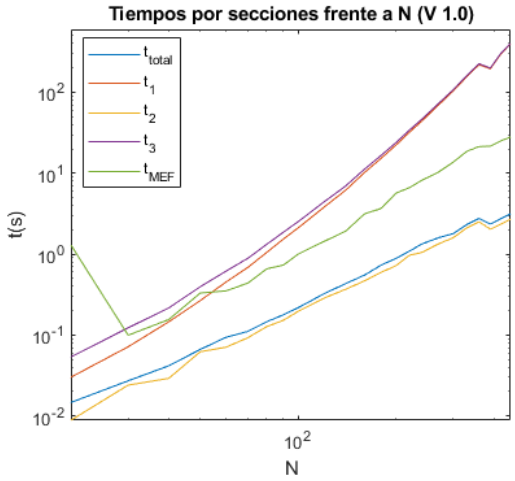

Figura 5.3 Tiempos de cálculo con mallados irregulares en V 1.0.

Tabla 5.4 Tiempos de cálculo por partes para red generada aleatoriamente.

N	20	80	160	240	330	450
Parte 1(s)	0.0147	0.1462	0.5504	1.3347	2.3189	3.1528
Parte 2(s)	0.0301	1.0479	10.2125	44.0317	156.0442	397.1398
Parte 3(s)	0.0089	0.1271	0.4672	1.0494	2.1089	2.6992
Total(s)	0.0537	1.3212	11.2301	46.4158	160.4720	402.9918
Solución MEF(s)	1.2973	0.6582	3.1478	8.1834	18.6325	27.9692

En ambas situaciones se tienen situaciones parecidas, los tiempos de la estimación del error es bastante superior al del cálculo de la solución aproximada. Ésto significa que el primer código, no es para nada válido para ser utilizado para mallados con muchos elementos, ya que, su rendimiento en ambos es muy bajo y tiene un gran margen de mejora *a priori*. Tener tiempos de cálculo más altos que los de elementos finitos es inaceptable, al estar reutilizando muchos datos . La aportación más grande al tiempo de cálculo, especialmente para mallados con muchos elementos, viene de la parte 2, en la que se resuelven los sistemas de ecuaciones. Según se ha planteado inicialmente, se resuelven n_N sistemas de ecuaciones, de modo que cuanto mayor sea la N mucho mayor es el número de nodos, y consecuentemente mayor es el número de matrices de coeficientes que hay que crear y de sistemas que se resuelven.

5.2 Versión 1.1

Esta versión nace con la siguiente pregunta, ¿ se pueden reducir el número de matrices de coeficientes que se crean? La respuesta es sí, debido a que siempre son iguales para los nodos que están en el mismo número de elementos y que son del mismo tipo, como se vio en la sección del Método del Residuo Equilibrado.

$$\begin{array}{rcl} AX_1 & = & b_1 \\ AX_2 & = & b_2 \\ & \vdots & \\ AX_n & = & b_n \end{array} \quad \equiv \quad A\left[\begin{array}{c|c|c|c} X_1 & X_2 & \dots & X_n \end{array}\right] = \left[\begin{array}{c|c|c|c} b_1 & b_2 & \dots & b_n \end{array}\right]$$

Luego un buen cambio en el planteamiento para la nueva versión, llamada versión 1.1, es ir resolviendo a la vez todos los sistemas de los nodos que están en un mismo número de triángulos. Siendo solamente necesario crear cuatro matrices de coeficientes, una para cada tipo de nodo, para cada número de triángulos en los que está un nodo, cifra que determina la dimensión de las matrices. El número de triángulos en los que esta cada nodo se puede conocer siguiendo el mismo procedimiento que se hacía para encontrar su valor máximo, mediante el comando **sparse**, quedando almacenado en el vector `NTri` siendo la componente i el número de triángulos en los que está el nodo de índice i. Además su valor máximo es el valor $NTri_{max}$, que determinará el tamaño de `sigmaden`, al igual que en la versión anterior.

```
NTri=sparse(T,ones(3*NT,1),ones(3*NT,1));
```

Introduciendo este idea al obtener la matriz `sigma`, se pasa de crear n_N veces las matrices de coeficientes y resolver su sistemas a $4NTri_{max}$, considerando que siempre hay al menos un nodo de cada tipo para cada número de triángulos en los que está un nodo. Generalmente, el valor máximo de `NTri` no tiene por qué ser mucho mayor en redes finas que en mallados poco refinados, sin embargo el número de nodos si es mucho mayor, luego este cambio debería aportar una reducción en el tiempo de cálculo para redes con un número cuantioso de nodos.

Con este planteamiento, al calcular $\sigma_{K,n}$ hay que hacer un bucle desde 1 hasta $NTri_{max}$, en el que se identifica los nodos que están el número de triángulos en cuestión mediante el comando **find**. Luego de esto, hay que separarlos por tipos. Para ello, se hace un bucle en por cada uno de ellos, identificándose de la misma forma que se hacía en la versión inicial, llenando unos vectores para cada tipo en los que se tienen el índice del nodo y unas matrices para los triángulos en los que están y el nodo que son cada uno dentro de ese triángulo. Además de tener en una matriz de la misma estructura con el valor de $\tilde{\Delta}_{K,n}$ correspondiente.

Después de ésto, se crean las matrices de coeficientes de cada tipo y se resuelven para todos los nodos iguales a la vez, aprovechando que la matriz de coeficientes es igual. La matriz de coeficientes de un tipo se crea siempre que haya al menos un nodo del mismo. Es decir, la solución de cada tipo será una matriz compuesta por columnas que contienen la solución de cada nodo. Por último, se asignan los valores de σ a su correspondiente posición en la matriz en la que se guarda $\sigma_{K,n}$, de la misma forma que se hacía en la versión 1.0 .

Código 5.1 Calculo de los multiplicadores de Lagrange.

```
%Calculo sigma
sigma=zeros(size(T));
NTri=sparse(T,ones(3*NT,1),ones(3*NT,1));
nmax=max(NTri);%
for j=1:nmax

    Nodosj=find(NTri==j); %nodos que estan en j triangulos

    NodosInt=zeros(length(Nodosj),1); %nodos interiores
    nInt=0;%numero de nodos
    TInt=zeros(j,length(Nodosj));% triangulos
    nkInt=TInt; %nodos de los triangulos
    deltaInt=TInt; %termino independiente

    NodosNeuNeu=NodosInt; %nodos neu-neu
    nNeuNeu=0;
    TNeuNeu=TInt;
    nkNeuNeu=TInt;
    deltaNeuNeu=TInt;

    NodosNeuDir=NodosInt;%nodos neu-dir
    nNeuDir=0;
    TNeuDir=TInt;
    nkNeuDir=TInt;
    deltaNeuDir=TInt;

    NodosDirDir=NodosInt; %nodos dir-dir
    nDirDir=0;
    TDirDir=TInt;
    nkDirDir=TInt;
    deltaDirDir=TInt;

    %tipo de nodo
    for n=Nodosj'
        [filas,columnas]=find(T==n);
        deltagorron=zeros(j,1);
        for k=1:j%vector con los indices de los triangulos
            deltagorron(k)=deltagorro(filas(k),columnas(k));
        end

        %es frontera?
        [~,frontera]=find(B==n);
        if isempty(frontera)
            %es interior
            nInt=nInt+1;
            NodosInt(nInt)=n;
            TInt(:,nInt)=columnas;
            nkInt(:,nInt)=filas;
```

```
            deltaInt(:,nInt)=deltagorron;

        else %es frontera, hay que ver el tipo
            [~,frontera]=find(ID==frontera);
            if isempty(frontera) %tipo N-N
                nNeuNeu=nNeuNeu+1;
                NodosNeuNeu(nNeuNeu)=n;
                TNeuNeu(:,nNeuNeu)=columnas;
                nkNeuNeu(:,nNeuNeu)=filas;
                deltaNeuNeu(:,nNeuNeu)=deltagorron;

            elseif length(frontera)==1 %tipo N-D
                nNeuDir=nNeuDir+1;
                NodosNeuDir(nNeuDir)=n;
                TNeuDir(:,nNeuDir)=columnas;
                nkNeuDir(:,nNeuDir)=filas;
                deltaNeuDir(:,nNeuDir)=deltagorron;

            else %tipo D-D
                nDirDir=nDirDir+1;
                NodosDirDir(nDirDir)=n;
                TDirDir(:,nDirDir)=columnas;
                nkDirDir(:,nDirDir)=filas;
                deltaDirDir(:,nDirDir)=deltagorron;
            end
        end
    end

    %Sistemas para los 4 tipos de nodo

    %Nodo interior
    if nInt>0
        d0=2*ones(1,j);
        d1=-1*ones(1,j-1);
        M=zeros(j+1,j);
        M(1:j,1:j)=diag(d0,0)+diag(d1,-1)+diag(d1,1);
        M(1,j)=-1;
        M(j,1)=-1;
        M(j+1,:)=ones(1,j);
        M=M/2;

        b=[deltaInt(:,1:nInt); zeros(1,nInt)];
        sigmaInt=M\b;

        %tenemos sigma, ahora hay que asignarlos al triangulo y nodo
        %correspondiente
        for p=1:nInt
            for k=1:j
                sigma(nkInt(k,p),TInt(k,p))=sigmaInt(k,p);
            end
```

```
    end
end

%nodos Neu-Neu
if nNeuNeu>0
    d0=[1 2*ones(1,j-2) 1];
    d1=-1*ones(1,j-1);
    M=zeros(j+1,j);
    if j==1
        M=1/2;
    else
        M(1:j,1:j)=diag(d0,0)+diag(d1,-1)+diag(d1,1);
        M(j+1,:)=ones(1,j);
        M=M/2;
    end

    b=[deltaNeuNeu(:,1:nNeuNeu); zeros(1,nNeuNeu)];
    sigmaNeuNeu=M\b;

    for p=1:nNeuNeu
        for k=1:j
            sigma(nkNeuNeu(k,p),TNeuNeu(k,p))=sigmaNeuNeu(k,p);
        end
    end
end

%nodos Neu-Dir

if nNeuDir>0
    d0=[3 2*ones(1,j-2) 1];
    d1=-1*ones(1,j-1);

    if j==1
        M=3/2;
    else
        M=diag(d0,0)+diag(d1,-1)+diag(d1,1);
        M=M/2;
    end

    b=deltaNeuDir(:,1:nNeuDir);

    sigmaNeuDir=M\b;

    for p=1:nNeuDir
        for k=1:j
            sigma(nkNeuDir(k,p),TNeuDir(k,p))=sigmaNeuDir(k,p);
        end
    end
end
```

```
    %nodos Dir-Dir

    if nDirDir>0

        M=zeros(j);
        if j==1
            M=3/2;
        else
            d0=[3 2*ones(1,j-2) 3];
            d1=-1*ones(1,j-1);
            M=diag(d0,0)+diag(d1,-1)+diag(d1,1);
            M=M/2;
        end

        b=deltaDirDir(:,1:nDirDir);

        sigmaDirDir=M\b;

        for p=1:nDirDir
            for k=1:j
                sigma(nkDirDir(k,p),TDirDir(k,p))=sigmaDirDir(k,p);
            end
        end
    end
end
```

Con el fin de ver la mejora que se obtiene con estos cambios se vuelven a obtener los tiempos de cálculo para poder realizar una comparación entre los de la parte 2 de la versión 1.0 y la 1.1. Los de las otras serán iguales debido a que su parte del código es idéntica, luego no tiene sentido incluirlas en la comparación.

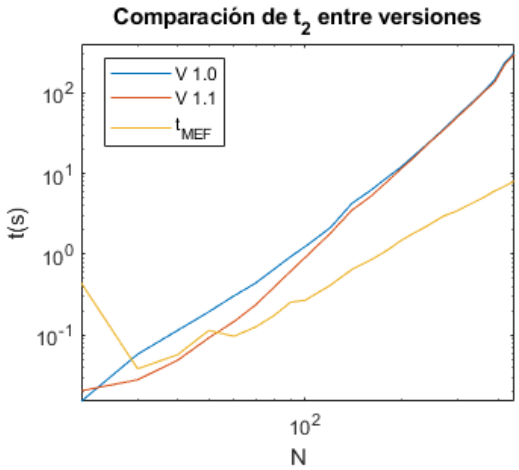

Figura 5.4 Comparación de tiempos de cálculo de la parte 2 con mallados regulares.

Observando los tiempos se ve que se tiene un tiempo algo menor en la mayoría los casos. De todas formas se ve que la diferencia no se hace mucho mayor para mallados más finos, consecuentemente los tiempos siguen siendo mucho mayores que los de elementos finitos. Esto se debe a que hay una gran cantidad de bucles para cada nodo y

Tabla 5.5 Comparación entre los tiempos de cálculo de la parte 2 para red regular.

N	20	80	160	240	330	450
V 1.0(s)	0.0152	0.6573	6.1568	22.9602	73.8507	311.7745
V 1.1(s)	0.0205	0.3929	5.1839	22.6283	71.7052	297.1738
Solución MEF(s)	0.4412	0.1751	0.8464	2.2335	4.2326	8.0890

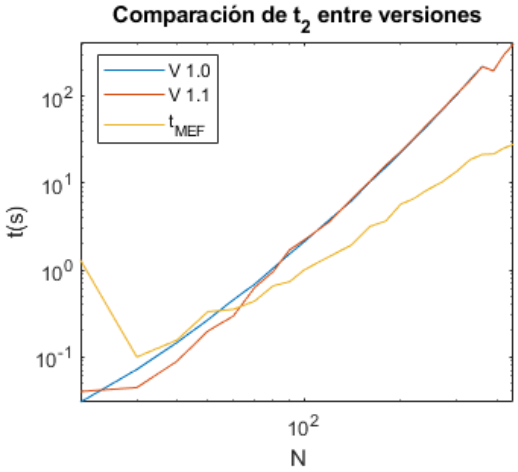

Figura 5.5 Comparación de tiempos de cálculo de la parte 2 con mallados irregulares.

Tabla 5.6 Comparación entre los tiempos de cálculo de la parte 2 para red generada aleatoriamente.

N	20	80	160	240	330	450
V 1.0(s)	0.0301	1.0479	10.2125	44.0317	156.0442	397.1398
V 1.1(s)	0.0399	0.9585	10.3899	45.4935	152.3175	395.9159
Solución MEF(s)	1.2973	0.6582	3.1478	8.1834	18.6325	27.9692

cada triángulo. Por lo que aún quedan muchos que ralentizan los cálculos aunque se haya reducido la influencia de uno de ellos. Como conclusión de esta modificación, se comprueba que resolver los sistemas de $\sigma_{K,n}$ de esta nueva forma consigue reducir los tiempos de cálculo, sin embargo éstos siguen siendo elevados y habrá que buscar nuevas modificaciones sobre el código.

5.3 Versión 1.2

Tras conocer que la forma en la que se resolvían los sistemas no es el principal problema que aumentaba los tiempos de calculo, se plantea reescribir todo el código eliminando todos los bucles posible, ya que, con en cada una de ellos se realizan las mismas operaciones una y otra vez. De esta forma se reduce el número de ordenes que tiene que ejecutar la herramienta de cálculo, siendo ésta la verdadera clave de la optimización. Al mismo tiempo, aquí es donde entra el uso de otras órdenes de MATLAB más complejas, que se describirán al ser usadas.

Para ir eliminando bucles se seguirá una estructura igual a la de la sección anterior, yendo por cada trozo del código y exponiendo los cambios.

1. Manipulación de los datos del Mallado. En este caso sólo se tenía un bucle para cada

triángulo con el que se obtenía la longitud de cada lado. Ésto se sustituye por dos ordenes en las que mediante el comando **diff** se obtienen las diferencias entre las matrices con las coordenadas x e y de los nodos se cada triángulos, `X` e `Y` de una forma igual a la de `T`, es decir, `X`(i,j) será la coordenada del nodo i del triángulo j, siendo igual en el caso de `Y`. Se crea la matriz `U`, con la misma estructura, en la que se tiene el valor de la solución en cada uno de los nodos de cada triángulo. Se puede obtener las longitudes con el comando **sqrt**, que admite matrices como entrada, utilizando el **.** para elevar al cuadrado todos los elementos de las matrices.
Además de esto, se calculan las componentes de las normales unitarias exteriores a cada lado, aprovechando que se tienen las longitudes de los lados y las matrices con las diferencias de las coordenadas.

Código 5.2 Datos del Mallado.

```
NT=length(T(1,:));%Numero de triangulos
Nn=length(z(1,:)); %Numero de nodos

X=z(1,:);
X=X(T);
Y=z(2,:);
Y=Y(T);
U=u(T);

Lx=diff(X([2 3 1 2],:));
Ly=diff(Y([2 3 1 2],:));
Ck=sqrt(Lx.^2+Ly.^2); %longitud de los lados

%normales
nx=Ly./Ck;
ny=-Lx./Ck;
```

En esta versión se incluyen también matrices con las que conocer el tipo de cada nodo y lado. En el caso de los nodos se utiliza un vector columna, llamado `TiposNodos`, de n_N componentes, una para cada nodo, similar al calculo de `NTri` mediante el comando **sparse**. Se da de entrada de filas una matriz creada por los índices de los nodos que son los extremos de los lados de frontera Neumann, seguidas de los indices de los de la frontera Dirichlet. Mientras que el vector de valores será un vector compuesto de dos partes , una con todas las componentes iguales a 0.5 la misma longitud que `IN` , y otra con las componentes de 1.5 con la longitud de `ID`. Sumando 0.5 a la fila de cada nodo al que llega un lado de frontera Neumann y 1.5 si lo hace uno de frontera Dirichlet.
Además se le introducen como entrada las dimensiones de la matriz que se quiere obtener $(n_N \times 1)$, en este caso.
De forma que se tienen los siguientes valores en función del tipo de nodo.

$$\texttt{TipoNodos}(n) = \begin{cases} 0 & \text{El nodo n es interior} \\ 1 = 0.5 + 0.5 & \text{El nodo n es Neumann-Neumann} \\ 2 = 0.5 + 1.5 & \text{El nodo n es Neumann-Dirichlet} \\ 3 = 1.5 + 1.5 & \text{El nodo n es Dirichlet-Dirichlet} \end{cases}$$

Código 5.3 Matriz de tipos de nodo.

```
NN=length(IN(1,:));
ND=length(ID(1,:));

TiposNodos=sparse([B(:,IN) B(:,ID)], ones(2,(NN+ND)), ...
    ...[0.5*ones(2,NN) 1.5*ones(2,ND)],Nn,1);
```

Para los tipos de los lados, se crea la matriz `TiposLados` de $(n_T \times 3)$, una fila para cada triángulo, y una columna por cada lado. Primero, mediante **find** se busca en `ite` los lados que son frontera, es decir, en los que $\texttt{ite}(i,j) < 0$. Con esto busca mediante **ismember** sobre los valores de `-ite`, con las filas y columnas que se han obtenido con el comando **find**, y el vector de lados de frontera Neumann, obteniéndose un vector auxiliar con ceros y unos, llamado `aux`. Siendo uno cuando el lado en cuestión está en el vector de frontera Neumann y 0 en el caso contrario. Cabe recalcar que a este comando hay que introducirle un vector, éste se obtiene gracias a que MATLAB almacena las matrices por columnas, es decir, las guarda como un vector columna. Luego para una matriz de NT filas, se puede decir que $A(i,j) = A((j-1)NT+i)$. Así se consiguen extraer a un vector las componentes en cuestión de `ite`. Finalmente se crea una matriz con el comando **sparse** que coloca las componentes del vector anterior en su correspondiente lugar de la matriz `TiposLados`, sumándole 2 a todas ellas con el fin de que en la matriz final sólo haya número del 0 al 2, como se verá después. Además se le resta 1 a todas las componentes y se le suma **sign(iteb)**, que da como salida una matriz del mismo tamaño que `iteb` con valores de 1 si el signo se esa componente es positivo, 0 si es nula y -1 si es negativa. Recordando que `iteb` es positiva para los lados interior y vale 0 en los lados exteriores.

Código 5.4 Matriz de tipos de lados.

```
[I,J]=find(ite<0);
aux=ismember(-ite((J-1)*NT+I),IN);
TiposLados=sign(iteb)-1+sparse(I,J,2+aux,NT,3);
```

Quedando los siguientes valores en función del tipo de lado.

$$\texttt{TipoLados}(K,l) = \begin{cases} 0 = 1-1 & \text{El lado l de K es interior} \\ 1 = -1+2 & \text{El lado l de K es frontera Dirichlet} \\ 2 = -1+3 & \text{El lado l de K es frontera Neumann} \end{cases}$$

Con estas matrices se consigue eliminar las comparaciones de cada nodo o cada lado para conocer su tipo. Como se verá más tarde, todas esas ordenes se pueden sustituir por el uso de **find** sobre estas matrices buscando el valor correspondiente.

2. Cálculo del gradiente. En las versiones anteriores se utilizaba un bucle para ir de elemento en elemento, que se puede eliminar perfectamente con el uso del calculo matricial siguiendo el mismo procedimiento. Para ello se utiliza **.** para multiplicar los componentes de las matrices uno a uno.

Código 5.5 Cambio de variables.

```
Nxi=[-1;1;0];
Neta=[-1;0;1];
```

```
denominador=((X(2,:)-X(1,:)).*(Y(3,:)-Y(1,:))-...
   ...(X(3,:)-X(1,:)).*(Y(2,:)-Y(1,:)));

xix=(Y(3,:)-Y(1,:))./denominador;

etax=-(Y(2,:)-Y(1,:))./denominador;

xiy=-(X(3,:)-X(1,:))./denominador;

etay=(X(2,:)-X(1,:))./denominador;
```

Además se usa el comando **kron**, que realiza el producto de Kronecker a los matrices dadas. Esta operación, sobre una matriz A $(m \times n)$ y otra B $(p \times q)$, obtiene una matriz de $(mp \times nq)$ con la siguiente relación:

$$A \otimes B = \begin{bmatrix} a_{11}B & \dots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \dots & a_{mn}B \end{bmatrix}$$

donde $\otimes$ denota producto de Kronecker. [5]
Con el producto de Kronecker sobre N_ξ y ξ_x, siendo ξ_x^T un vector con un valor columna con una componente para cada elemento, se tiene una matriz de $(3 \times n_T)$ de la siguiente forma:

$$N_\xi \otimes \xi_x^T = \begin{bmatrix} -\xi_x(1) & -\xi_x(2) & \dots & -\xi_x(n_T) \\ \xi_x(1) & \xi_x(2) & \dots & \xi_x(n_T) \\ 0 & 0 & \dots & 0 \end{bmatrix}$$

Luego si se hace un sumatorio de todas sus filas, se llega a un vector de n_T componentes que sería equivalente al producto que se hacía en cada bucle. Este razonamiento se extrapola para los otros términos con los que se calcula el gradiente. Llegando a las siguientes líneas de código para calcular el gradiente sin usar ningún bucle.

Código 5.6 Obtención del gradiente.

```
gradux=zeros(2,NT);
gradux(1,:)=sum(U.*(kron(Nxi,xix(:)')+kron(Neta,etax(:)')));
gradux(2,:)=sum(U.*(kron(Nxi,xiy(:)')+kron(Neta,etay(:)')));
```

3. Obtención del momento del flujo aproximado, $\tilde{\mu}_{K,n}^{\gamma}$. Este paso se realizaba con dos bucles, uno para cada triángulo y otro para cada lado de los mismos, luego conseguir eliminarlos sería muy ventajoso. Para conseguir hacerlo , en primer lugar se guarda en una matriz $(3 \times n_T)$ el producto escalar entre la normal exterior a cada lado y el gradiente de la solución aproximada en ese elemento mediante al producto elemento a elemento entre ambos. Siendo la componente (i,j) de esa matriz el producto escalar en el elemento j entre su gradiente y la normal exterior al lado (i,j).
Para calcular el flujo se utiliza el comando **repmat** con el que se pueden crear matrices a raíz de repetir las filas y columnas de una matriz o un vector. Utilizando esto sobre la matriz de las longitudes, expresada como un vector fila, la matriz de los productos escalares, de igual modo que la anterior, y repitiendo n_T veces la matriz `IntThetan` , se

llega a poder calcular de la misma forma de la que se hacía lado a lado. Veáse el ejemplo para un triángulo K, tomando las submatrices correspondientes.

$$\tilde{\mu}^{\gamma}_{K,n} = \int_{L_j} n^j_K \cdot \nabla u_x|_K \theta_i \, ds =$$

$$= \begin{bmatrix} n^1_K \cdot \nabla u_x|_K & n^2_K \cdot \nabla u_x|_K & n^3_K \cdot \nabla u_x|_K \\ n^1_K \cdot \nabla u_x|_K & n^2_K \cdot \nabla u_x|_K & n^3_K \cdot \nabla u_x|_K \\ n^1_K \cdot \nabla u_x|_K & n^2_K \cdot \nabla u_x|_K & n^3_K \cdot \nabla u_x|_K \end{bmatrix} .* \begin{bmatrix} C_1|_K & C_2|_K & C_3|_K \\ C_1|_K & C_2|_K & C_3|_K \\ C_1|_K & C_2|_K & C_3|_K \end{bmatrix} .* \begin{bmatrix} 0 & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} & 0 \end{bmatrix}$$

donde $.*$ denota el producto elemento a elemento. En la práctica se realiza ese cálculo a la vez para todos los triángulos, ya que las matrices que se utilizan son de $(3 \times 3n_T)$, es decir, están compuestas por n_T submatrices con esa forma.

Código 5.7 Cálculo del momento del flujo aproximado.

```
IntThetan=[0 0.5 0.5; 0.5 0 0.5; 0.5 0.5 0 ]'; %En eta xi
gradun=gradux(1,:).*nx+gradux(2,:).*ny; %esta bien
muAprox=repmat(Ck(:)',3,1).*repmat(IntThetan,1,NT).*...
   ...repmat(gradun(:)',3,1);
```

4. Calculo de $\tilde{\Delta}_{K,n}$. Recordando la ecuación de este término:

$$\tilde{\Delta}_K(\theta_n) = B_K(u_x,\theta_n) - (f,\theta_n)_K - \int_{\partial K} \left\langle \frac{\partial u_x}{\partial n_K} \right\rangle \theta_n \, ds$$

Al calcular sus dos primeros términos se usaba un bucle para ir elemento a elemento, que se elimina fácilmente en el caso de $(f,\theta_n)_K$, ya que, en el fondo sólo se creaba una matriz igual a `fe`, que es dato del problema. Mientras que el caso de $B_K(u_x,\theta_n)$ tiene algo de complejidad. Originalmente se hacia el producto de la submatriz de `Ae` ,correspondiente a cada elemento, con el vector con el valor de u_x en sus nodos. Para poder hacer esta multiplicación matricial simultáneamente en todos los triángulos se sabe que esta operación se compone de multiplicar elemento a elemento cada fila de $\texttt{Ae}|_K$ con $u_x|_K$ y después sumarlos, de forma que quedan asignados en la fila correspondiente de $\texttt{Ae}|_K$. Luego si se consigue hacer el primer paso con una multiplicación elemento a elemento sólo habría que realizar un sumatorio de filas. Con este razonamiento se crea una matriz $(3n_T \times 3)$ compuesta por n_T matrices que tienen 3 filas repetidas, iguales a $u_x|_K$, mediante el uso de **kron** sobre `U`, matriz con valores nodales de u_x, y un vector de unos. De esta forma, se tiene una matriz con la que multiplicar elemento a elemento con `Ae`. Después quedaría hacer un sumatorio de las filas y se tendría un vector de $3n_T$ componentes, compuesto por n_T vectores que contienen $[B_K(u_x,\theta_1),B_K(u_x,\theta_2),B_K(u_x,\theta_3)]^T$. Véase a continuación el ejemplo de ésto en triángulo K.

$$\begin{bmatrix} B_K(u_x,\theta_1) \\ B_K(u_x,\theta_2) \\ B_K(u_x,\theta_3) \end{bmatrix} = \begin{bmatrix} Ae_{11}|_K & Ae_{12}|_K & Ae_{13}|_K \\ Ae_{21}|_K & Ae_{22}|_K & Ae_{33}|_K \\ Ae_{31}|_K & Ae_{32}|_K & Ae_{33}|_K \end{bmatrix} .* \begin{bmatrix} u_1 & u_2 & u_3 \\ u_1 & u_2 & u_3 \\ u_1 & u_2 & u_3 \end{bmatrix}$$

Este vector se transforma en una matriz `Bk` como la que se tenía en versiones anteriores mediante el comando **reshape**, que va llenando una matriz de dimensiones dadas con las componentes de un vector dado.

```
fek=fe;

Bk=reshape(sum(Ae.*kron(U',ones(3,1)),2),3,NT);
```

Ahora sólo queda el último término, que como se vio anteriormente depende del tipo de cala lado. Antes se calculada con dos bucles y dentro de ellos se utilizaban varios **if**, por lo que eliminarlos podría reducir los tiempos de cálculo cuando hay un grande número de elementos. Aquí también se consiguen eliminar todos los bucles, con la particularidad de que se tienen que realizar 3 cálculos diferentes, uno para cada tipo de lado. Es decir, para todos los lados del mismo tipo se calcula simultáneamente, siendo necesario obtener la información para definir todos los lados cada tipo. La matriz `TiposLados` se ha creado para eso, mediante la búsqueda del valor asignado a cada tipo en esa matriz con **find**, se obtiene `I`, un vector con los triángulos en los que está cada lado, y `J`, vector con el lado que es en cada triángulo que se encuentra en ese lugar de `I`.
En el caso del lado interior, hay que realizar una semisuma entre el vector de momentos aproximados del flujo de cada lado con el opuesto de su triángulo vecino en cada uno de ellos. Se hará primero la parte del triángulo en cuestión y después se le añadirá la aportación del vecino. En primer lugar habrá que buscar en `TiposLados` el valor 0, consiguiendo los correspondientes `I` y `J`. Con éstos se crea una matriz auxiliar compuesta por los momentos aproximados, que son vectores columna de 3 componentes, referentes a los lados que definen los vectores `I` y `J`. Se utiliza **sparse** asignar estos valores en su posición correspondiente de `Intuxnk`, matriz $(3 \times n_T)$en la que se que almacena los valores de la integral para cada triángulo. La razón para elegir este comando es la misma que cuando se generó la matriz `NTri`, porque al querer sumar todos los lados de cada triángulo en una misma columna, si se da un valor a una componente que ya tiene, éstas se suman. Para asignar los valores almacenados en la matriz auxiliar multiplicados por 0.5, sólo habrá que crear la matriz de filas del mismo tamaño, compuesta por la repetición de estos vectores $[1,2,3]^T$, que indican a las funciones base que se han tomado para calcular cada momento; y otra matriz con las columnas, que será la repetición 3 veces del vector `I`.

Código 5.8 Primera parte de la integral del producto escalar en los lados internos.

```
[I,J]=find(TiposLados==0);
aux=muAprox(:,(I-1)*3+J);
fila=repmat((1:3)',1,length(aux(1,:)));
columna=repmat(I',3,1);
Intuxnk=0.5*sparse(fila,columna,aux,3,NT);
```

En estos tipos de lados todavía queda incluir la aportación de los triángulos vecinos, éstos y sus correspondientes lados quedarán definidos por los valores de `I` y `J` en la matriz `ite` e `iteb`. Se extrae los vectores `Ivec` y `Jvec` extrayendo esos datos de las matrices citadas. Con éstos datos se repite el proceso hecho anteriormente, con la particularidad que el signo de la matriz auxiliar será negativo y de que hay que ordenar el orden de los índices de los nodos, para que coincidan en los dos triángulos colindantes, mediante el uso de la matriz `Ordenacion`. Es decir, la matriz que determina las filas en las que se asignan los valores de la matriz auxiliar serán columnas

de `Ordenacion`, elegidas en función de cuál sea el valor del lado y lado vecino, almacenados en `J` y `Jvec` respectivamente. Con todo ello se suma la nueva matriz dispersa a la que se tenía de antes.

Código 5.9 Segunda parte de la integral del producto escalar en los lados internos.

```
          %L1 de k   %L2 de k  %L3 de k
Ordenacion=[ 1 2 3    2 3 1     3 1 2; %theta 2 de k
             3 1 2    1 2 3     2 3 1; %theta 2 de k
             2 3 1    3 1 2     1 2 3];%theta 3 de k
          %L1,L2,L3 de k
Ivec=ite((J-1)*NT+I);
Jvec=iteb((J-1)*NT+I);
aux=muAprox(:,(Ivec-1)*3+Jvec);
%hay que cambiar el orden
fila=Ordenacion(:,3*(J-1)+Jvec);
Intuxnk=Intuxnk-0.5*sparse(fila,columna,aux,3,NT);
```

Para el caso de los Dirichlet, sería prácticamente igual que la primera parte del caso anterior, con la diferencia de que se busca el valor 1 en la matriz `TiposLados` y que la matriz auxiliar no se multiplica por 0.5.

Código 5.10 Integral del producto escalar en los lados Dirichlet.

```
[I,J]=find(TiposLados==1);
aux=muAprox(:,(I-1)*3+J);
fila=repmat((1:3)',1,length(aux(1,:)));
columna=repmat(I',3,1);
Intuxnk=Intuxnk+sparse(fila,columna,aux,3,NT);
```

Sólo queda añadir la aportación de los lados de frontera Neumann. Se busca en la matriz `TiposLados` el valor 2, la matriz auxiliar será la matriz `ge` a la que se le añade una fila de ceros. Para poder asignar los valores a sus correspondientes funciones nodales y elemento hace falta que el orden de de los vectores `I` y `J` coherentes a como están dispuestos los datos en `ge`. Primero se introduce la matriz `OrdenLNeu`, cuya función es la equivalente a la de `Ordenacion`, ordenando las componentes de `ge` en función del índice de cada lado.

$$\texttt{OrdenLNeu} = \begin{bmatrix} 2 & 3 & 1 \\ 3 & 1 & 2 \\ 1 & 2 & 3 \end{bmatrix}$$

En la matriz `ge` los datos están dispuestos por columnas desde el lado de frontera Neumann con menor indice hasta el de mayor. Para conseguir que `I` y `J` estén dispuestos de esa forma, se extraen estos índices en el vector `LNeu` de la matriz `ite` los índices de lado de frontera asignados a cada valor de esta pareja de vectores. Después se obtiene el vector que los ordena mediante el comando **sort** sobre `LNeu`. Finalmente se repite el proceso del caso anterior, estando `I` y `J` re-ordenados. Y finalmente, se obtiene el valor de $\tilde{\Delta_{K,n}}$ sumando sus tres términos.

Código 5.11 Integral del producto escalar en los lados Neumann.

```
[I,J]=find(TiposLados==2);
ge=[ge;zeros(1,length(ge(1,:)))];
OrdenLNeu=[2  3   1;
           3   1   2;   %Componente de ge
           1   2   3];
          %L1  L2  L3
LNeu=-ite((J-1)*NT+I);
[~,orden]=sort(LNeu);
aux=ge;
fila=OrdenLNeu(:,J(orden));
columna=repmat(I(orden)',3,1);
Intuxnk=full(Intuxnk+sparse(fila,columna,aux,3,NT));


deltagorro=Bk-fek-Intuxnk;
```

5. Obtención de los multiplicadores de Langrange $\sigma_{K,n}$. Aquí es donde se gasta el mayor tiempo de cálculo para mallados finos, como se vio anteriormente, así que reducir las operaciones que realiza MATLAB resulta muy beneficioso para los tiempos. Al igual que con los lados, se calculará para cada tipo de nodos, siendo este el único paso en el que no se pueden eliminar todos los bucles. Ésto se debe a que realizar un bucle para crear las matrices de coeficientes correspondientes es imprescindible.
   Lo primero será crear un vector que contiene los índices de los nodos para cada tipo mediante el uso de **find** sobre la matriz `TiposNodos`, buscando su valor correspondiente. Antes de comenzar a ir resolviendo por cada tipo de nodo, se crean unas matrices con una información complementaria a `NTri` en cuya fila j se guardan los triángulos en los que está el nodo j y su índice en éstos. Ambas son matrices dispersas con posiciones vacías, luego para obtener un vector referente a un nodo con componentes no nulas se puede utilizar el comando **nonzeros**.

Código 5.12 Clasificación de los nodos.

```
NodosInt=find(TiposNodos(:,1)==0)';
NodosNeuNeu=find(TiposNodos(:,1)==1)';
NodosNeuDir=find(TiposNodos(:,1)==2)';
NodosDirDir=find(TiposNodos(:,1)==3)';

sigma=zeros(3,NT);
NTri=sparse(T,ones(3*NT,1),ones(3*NT,1));
nmax=max(NTri);% Numero mas alto de triangulos en los que puede estar
     un nodo

%triangulos en los que esta cada nodo
TNodos=sparse(T,repmat(1:NT,3,1),repmat(1:NT,3,1));
%indices en los triangulos
NNodos=sparse(T,repmat(1:NT,3,1),repmat((1:3)',1,NT));
```

Con esto ya sí se puede comenzar a resolver los sistemas. Se empieza con los nodos interiores. Se hace un bucle que determina el número de triángulos en los que va a estar los nodos que se tomen, en este caso se empieza por un valor de 3 porque es imposible

que un nodo interior esté en un número menor. Geométricamente, todos los ángulos internos de un triángulo debe de sumar 180º, entonces al menos serán necesarios 3 para llevar a completar 360º. Para cada vuelta en el bucle, se buscan los nodos interiores que están en el número j de triángulos y sus índices se guardan en el vector `Nodosj`. Si hay al menos un nodo, se sigue el mismo proceso que en la versión anterior, pero con otros órdenes al obtenerse todos los datos de los nodos al mismo tiempo. Ahora, se extraen los valores de los triángulos en los que está cada nodo y sus índices en ellos directamente de las correspondientes filas de las matrices `TNodos` y `NNodos`, a lo que se le suma el uso de **nonzeros**. Siendo el resultado en cada matriz es un vector de que se contiene de varios subvectores de j componentes que contienen la información de cada nodo. Entonces para tenerlos de una forma igual a la de versiones anteriores se vuelve a utilizar **reshape**. Con los términos independientes pasa exactamente lo mismo. El siguiente paso es construir la matriz correspondiente a este tipo de nodo y a j y resolverlo , obteniéndose la matriz `sigmaj`. Finalmente, se introducen los valores en su posición correcta de `sigma` igual que en la versión 1.1.

Código 5.13 Cálculo de los multiplicadores de Lagrange en nodos interiores.

```
for j=3:nmax %nodos interiores que estan en j triangulos
    Nodosj=find(NTri(NodosInt)==j); %nodos que estan en j triangulos

    nj=length(Nodosj);

    if nj>0
        Nodosj=NodosInt(Nodosj);%indices

        Tj=reshape(nonzeros(TNodos(Nodosj,:)'),j,length(Nodosj));
        nkj=reshape(nonzeros(NNodos(Nodosj,:)'),j,length(Nodosj));
        deltaj=reshape(deltagorro(nkj+3*(Tj-1)),j,length(Nodosj));

        d0=2*ones(1,j);
        d1=-1*ones(1,j-1);
        M=zeros(j+1,j);
        M(1:j,1:j)=diag(d0,0)+diag(d1,-1)+diag(d1,1);
        M(1,j)=-1;
        M(j,1)=-1;
        M(j+1,:)=ones(1,j);
        M=M/2;

        b=[deltaj(:,1:nj); zeros(1,nj)];
        sigmaj=M\b;

        sigma(nkj+(Tj-1)*3)=sigmaj;
    end
end
```

Para los otros tipos de nodos sería un proceso similar, cambiando el vector de nodos sobre el que se busca, la matriz de coeficientes y haciendo que j empiece desde 1, dado que ese caso se da en los nodos que se encuentran en las esquinas del dominio.

Código 5.14 Cálculo de los multiplicadores de Lagrange en nodos fronterizos.

```
for j=1:nmax

    Nodosj=find(NTri(NodosNeuNeu)==j);

    nj=length(Nodosj);

    if nj>0
        Nodosj=NodosNeuNeu(Nodosj);

        Tj=reshape(nonzeros(TNodos(Nodosj,:)'),j,length(Nodosj));
        nkj=reshape(nonzeros(NNodos(Nodosj,:)'),j,length(Nodosj));
        deltaj=reshape(deltagorro(nkj+3*(Tj-1)),j,length(Nodosj));

        d0=[1 2*ones(1,j-2) 1];
        d1=-1*ones(1,j-1);
        M=zeros(j+1,j);
        if j==1
            M=1/2;
            sigmaj=M\deltaj;
        else
            M(1:j,1:j)=diag(d0,0)+diag(d1,-1)+diag(d1,1);
            M(j+1,:)=ones(1,j);
            M=M/2;
            b=[deltaj(:,1:nj); zeros(1,nj)];
            sigmaj=M\b;
        end

        sigma(nkj+(Tj-1)*3)=sigmaj;
    end
end

for j=1:nmax

    Nodosj=find(NTri(NodosNeuDir)==j);

    nj=length(Nodosj);

    if nj>0
        Nodosj=NodosNeuDir(Nodosj);

        Tj=reshape(nonzeros(TNodos(Nodosj,:)'),j,length(Nodosj));
        nkj=reshape(nonzeros(NNodos(Nodosj,:)'),j,length(Nodosj));
        deltaj=reshape(deltagorro(nkj+3*(Tj-1)),j,length(Nodosj));

        d0=[3 2*ones(1,j-2) 1];
        d1=-1*ones(1,j-1);
```

```
        if j==1
            M=3/2;
        else
            M=diag(d0,0)+diag(d1,-1)+diag(d1,1);
            M=M/2;
        end

        b=deltaj(:,1:nj);
        sigmaj=M\b;

        sigma(nkj+(Tj-1)*3)=sigmaj;
    end
end

for j=1:nmax

    Nodosj=find(NTri(NodosDirDir)==j);

    nj=length(Nodosj);

    if nj>0
        Nodosj=NodosDirDir(Nodosj);

        Tj=reshape(nonzeros(TNodos(Nodosj,:)'),j,length(Nodosj));
        nkj=reshape(nonzeros(NNodos(Nodosj,:)'),j,length(Nodosj));
        deltaj=reshape(deltagorro(nkj+3*(Tj-1)),j,length(Nodosj));


        if j==1
            M=3/2;
        else
            d0=[3 2*ones(1,j-2) 3];
            d1=-1*ones(1,j-1);
            M=diag(d0,0)+diag(d1,-1)+diag(d1,1);
            M=M/2;
        end

        b=deltaj(:,1:nj);
        sigmaj=M\b;

        sigma(nkj+(Tj-1)*3)=sigmaj;
    end
end
```

6. Construcción de los momentos de los flujos, $\mu^{\gamma}_{K,n}$. Siguiendo un procedimiento equivalente al del cálculo de los flujos aproximados usando los vectores `I` y `J` de cada caso, añadiendo la influencia de los multiplicadores de Langrange en los lados interiores y Dirichlet.

Código 5.15 Cálculo de los momentos del flujo.

```
mu=zeros(3,3*NT);

[I,J]=find(TiposLados==0);
aux=sigma(:,I)+muAprox(:,3*(I-1)+J);
fila=repmat((1:3)',1,length(aux(1,:)));
columna=repmat((3*(I-1)+J)',3,1);
mu(fila+3*(columna-1))=0.5*aux;

Ivec=ite((J-1)*NT+I);
Jvec=iteb((J-1)*NT+I);
aux=sigma(:,Ivec)+muAprox(:,3*(Ivec-1)+Jvec);

fila=Ordenacion(:,3*(J-1)+Jvec);
mu(fila+3*(columna-1))=mu(fila+3*(columna-1))-0.5*aux;


[I,J]=find(TiposLados==1);
aux=sigma(:,I)+muAprox(:,3*(I-1)+J);
fila=repmat((1:3)',1,length(aux(1,:)));
columna=repmat((3*(I-1)+J)',3,1);
mu(fila+3*(columna-1))=aux;

LNeu=-ite((J-1)*NT+I);
[~,orden]=sort(LNeu);
aux=ge;
I=I(orden);
J=J(orden);
fila=OrdenLNeu(:,J);
columna=repmat((3*(I-1)+J)',3,1);
mu(fila+3*(columna-1))=aux;
```

7. Integral del flujo sobre los lados. Cuando se reconstruyen los valores del flujo para los extremos del flujo se consigue eliminar los bucles para cada triángulo y lado. Aquí se crean matrices repetidas con los números de los nodos iniciales y finales de cada lado, igual que el caso anterior, pero remetidas n_T veces para que se pueda calcular en todos los triángulos a la vez. Al igual que con el cálculo de $\tilde{\mu}^{\gamma}_{K,n}$, se repite una matriz con las longitudes de los lados, con las que dividir elemento a elemento en este caso. De forma que se realiza este cálculo para todos los triángulos con una misma operación:

$$\begin{bmatrix} g_l|_K \\ g_2|_K \end{bmatrix} = 2 \begin{bmatrix} 2\mu^1_{K,2} - \mu^1_{K,3} & 2\mu^2_{K,3} - \mu^2_{K,1} & 2\mu^3_{K,1} - \mu^3_{K,2} \\ -\mu^1_{K,2} + 2\mu^1_{K,3} & -\mu^2_{K,3} + 2\mu^2_{K,1} & -\mu^3_{K,1} + 2\mu^3_{K,2} \end{bmatrix} ./ \begin{bmatrix} C_1|_K & C_2|_K & C_3|_K \\ C_1|_K & C_2|_K & C_3|_K \end{bmatrix}$$

donde ./ denota la división elemento a elemento.

Código 5.16 Reconstrucción del flujo.

```
ThetaInicial=repmat([2,3,1],1,NT);
ThetaFinal=repmat([3,1,2],1,NT);
```

```
gk=2./repmat(Ck(:)',2,1).*[2*mu(3*(0:3*NT-1)+ThetaInicial)-...
   ...mu(3*(0:3*NT-1)+ThetaFinal);
    -mu(3*(0:3*NT-1)+ThetaInicial)+2*mu(3*(0:3*NT-1)+ThetaFinal)];
```

Por último se calculan todas las integrales sobre todos los lados. Recordando que `gk` es una matriz de $(2 \times 3n_T)$, en la que las columnas 1,4,7,10... están referidas a los lados 1 de todos los triángulos, las 2,5,8,11... a los lados 2 y las 3,6,9,12... a los lados 3. Entonces se puede calcular la integral en cada uno de ellos como la mitad de la suma del producto elemento a elemento del vector con la media del flujo para los lados 1 con el vector con las longitudes de los lados 1, que sería la primera fila de `Ck`. Quedando el valor de la integral en todo el contorno del elemento como la suma de todos los lados .

Código 5.17 Integral del flujo del flujo.

```
Intgk=0.5*(sum(gk(:,1:3:3*NT-2)).*Ck(1,:)+...
  ...sum(gk(:,2:3:3*NT-1)).*Ck(2,:)+sum(gk(:,3:3:3*NT)).*Ck(3,:));
```

Como se ha visto se han se ha pasado de tener varios bucles en cada parte a únicamente tener 3, barriendo desde 1 o 3 hasta valores que generalmente serán a lo sumo del orden de 15 o 20, mucho menor que la gran cantidad de bucles que corrían decenas de miles de veces con mellados muy refinados. Para ver si la mejora que se obtiene es importante se volverán a medir los tiempos de cálculo y se compararán entre las distintas versiones.

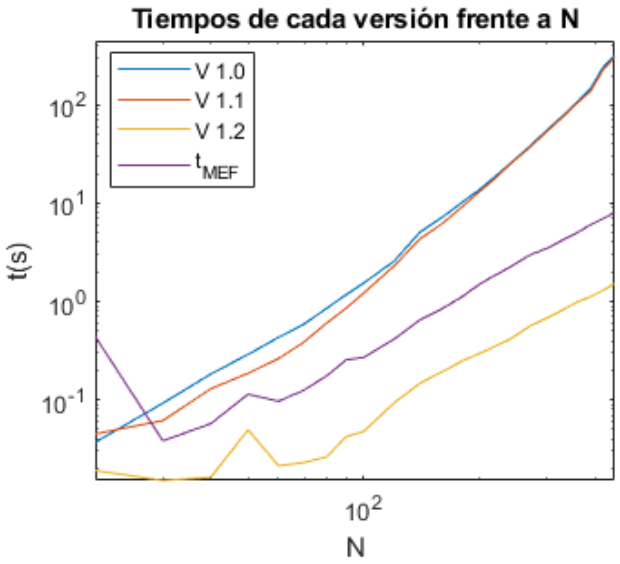

Figura 5.6 Comparación de tiempos con mallados regulares.

Tabla 5.7 Comparación entre los tiempos totales de cálculo para redes regulares.

N	20	80	160	240	330	450
V 1.0(s)	0.0152	0.6573	6.1568	22.9602	73.8507	311.7745
V 1.1(s)	0.0205	0.3929	5.1839	22.6283	71.7052	297.1738
V 1.2(s)	0.0191	0.0263	0.1931	0.4125	0.8232	1.5173
Solución MEF(s)	0.4412	0.1751	0.8464	2.2335	4.2326	8.0890

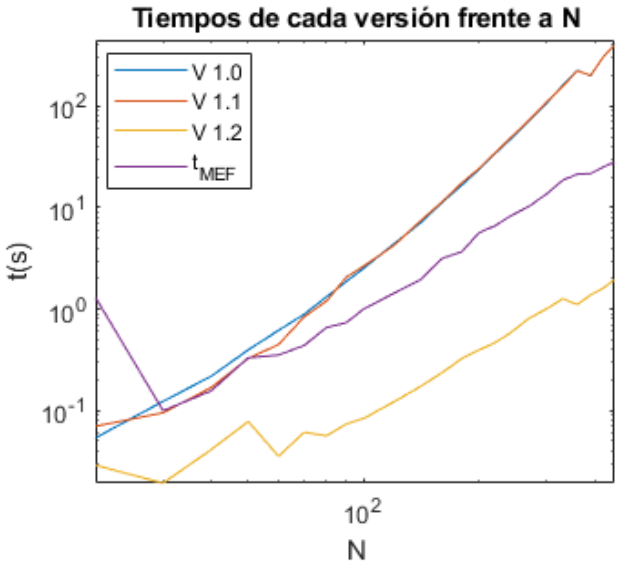

Figura 5.7 Comparación de tiempos con mallados irregulares.

Tabla 5.8 Comparación entre los tiempos totales de cálculo para redes generadas aleatoriamente.

N	20	80	160	240	330	450
V 1.0(s)	0.0301	1.0479	10.2125	44.0317	156.0442	397.1398
V 1.1(s)	0.0399	0.9585	10.3899	45.4935	152.3175	395.9159
V 1.2(s)	0.0288	0.0562	0.2357	0.5745	1.2625	1.9386
Solución MEF(s)	1.2973	0.6582	3.1478	8.1834	18.6325	27.9692

En relación a la comparación con las otras versiones se observa una diferencia mayúscula. Por ejemplo, en el último caso de la red con elementos irregulares anteriormente se tenían tiempos superiores a los 6 minutos, y con la nueva versión el tiempo no supera los 2 segundos.

Con todos estos datos se observa que la relación entre ambos tiempos crece para mallado refinado, es decir, cuando más grande sea el tiempo de cálculo más se nota que los tiempos de cálculos de la versión final es mucho menor que los tiempos de cálculo de la solución, por lo que cuando se calcule una simultáneamente al otro, éste será un orden de magnitud menor que el de elementos finitos, en la mayoría de los casos. Estos tiempos ya sí son coherentes con la idea de la reutilización de valores, es decir, con estos tiempos ya se puede consideran que se están aprovechando correctamente.

Esta mejora es muy importante, para entenderla mejor se expondrán los datos de la aportación al tiempo total de cada parte del código.

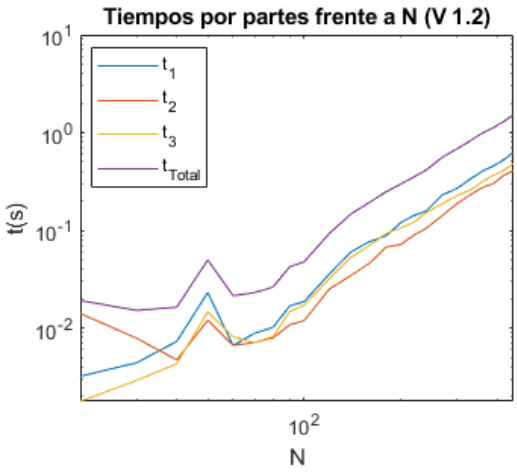

Figura 5.8 Tiempos por partes con mallados regulares en V 1.2.

Tabla 5.9 Tiempos por partes de la Versión 1.2 para red regular.

N	20	80	160	240	330	450
Parte 1(s)	0.0032	0.0102	0.0769	0.1567	0.3349	0.6285
Parte 2(s)	0.0141	0.0079	0.0462	0.1045	0.2303	0.4081
Parte 3(s)	0.0018	0.0082	0.0700	0.1513	0.2580	0.4807
Total(s)	0.0191	0.0263	0.1931	0.4125	0.8232	1.5173

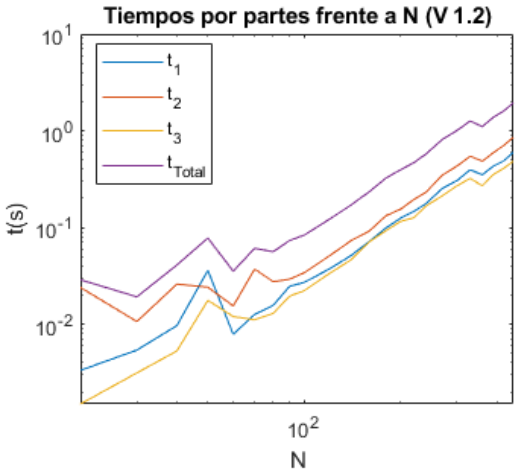

Figura 5.9 Tiempos por partes con mallados irregulares en V 1.2.

La distribución de tiempos difiere con los anteriores, en los que prácticamente todo el tiempo se dedicaba a obtener $\sigma_{K,n}$. En este caso, los tiempos son parecidos en todas las partes, siendo algo mayor en la parte 2 debido a que en ésta hay un bucle en el que en cada iteración se tienen que crear matrices y resolver hasta 4 sistemas.

Tabla 5.10 Tiempos por partes de la Versión 1.2 para redes irregulares.

N	20	80	160	240	330	450
Parte 1(s)	0.0033	0.0157	0.0720	0.1764	0.3955	0.5996
Parte 2(s)	0.0240	0.0275	0.0924	0.2315	0.5453	0.8553
Parte 3(s)	0.0015	0.0130	0.0713	0.1667	0.3217	0.4837
Total(s)	0.0288	0.0562	0.2357	0.5745	1.2625	1.9386

6 Conclusiones

Unas de las primeras ideas que se tiene generalmente sobre la optimización es que se basa en ideas enrevesadas que surgen de lo que se conoce como "idea feliz". Sin embargo, tras realizar toda este estudio sobre la implementación del Método del Residuo Equilibrado se llega a comprender que puede llegar a todo lo contrario, siempre que se sepa lo que se quiere llegar a hacer. Es decir, la optimización llega a parecer algo lógico con las bases de conocimiento sobre el aspecto en cuestión. El proceso bastará con estudiar la influencia de ciertos factores sobre el valor a optimizar. Tras hacer eso se busca la forma de conseguir reducir la existencia de los que afectan negativamente, como en este caso el uso de bucles. Para llevarlo a cabo en cada uno de los pasos del proceso, se intenta reducir cada pequeña parte en ideas simples que se pueden llegar a construir de diferentes formas, tomando la que mejor afecta a la solución final.

Finalmente cuando se ha realizado este profundo análisis sobre todo, y se han hecho los cambios, se llega a una solución óptima. Ésta suele ser concisa y sobria, como en este caso, en el que se ha pasado de calcular todo mediante bucles elemento a elemento al uso del operaciones directas con matrices con el uso de unos pocos comandos de MATLAB. Con el resultado de un código que comprime todo lo posible las ideas, hasta tal punto de llegar a parecer sencillas.

En definitiva, la optimización es un fruto del conocimiento que no sólo mejora los resultados, sino que además simplifica la forma de la solución de una forma elegante.

Índice de Figuras

Índice de Tablas

Índice de Códigos

Bibliografía

[1] Federico París, "*Capítulo 12: El Método de los Elementos Finitos*", *Teoría de la Elasticidad*, 1996.

[2] Endre Süli, *Lecture Notes on Finite Element Methods for Partial Differential Equations*, 2019.

[3] WIKIPEDIA , *Método de los Elementos Finitos*. [Consulta: 20 de julio 2021]. [**Consultarla**].

[4] Mark Ainsworth, J.Tinsley Oden, *A posteriori error estimation in finite element analysis*, 1997.

[5] WIKIPEDIA , *Producto de Kronecker*. [Consulta: 26 de julio 2021]. [**Consultarla**].