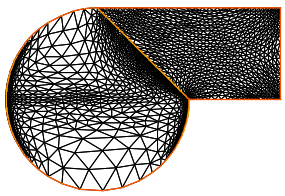
UNIVERSIDAD DE SALAMANCA

Facultad de Ciencias

Departamento de Matemática Aplicada

"Métodos de descomposición de dominio con adaptación de mallado en problemas de Convección-Reacción-Difusión"

Tesis Doctoral



Maria Manuela André Alves Simões

Salamanca, 2010

UNIVERSIDAD DE SALAMANCA

Facultad de Ciencias
Departamento de Matemática Aplicada

"Métodos de descomposición de dominio con adaptación de mallado en problemas de Convección-Reacción-Difusión"

Maria Manuela André Alves Simões

Memoria para optar al grado de Doctor en Matemáticas, realizada bajo la dirección de Dr. D. Luis Ferragut Canals del Departamento de Matemática Aplicada de la Universidad de Salamanca.

Salamanca, 30 de Septiembre de 2010

UNIVERSIDAD DE SALAMANCA

Facultad de Ciencias

Departamento de Matemática Aplicada

Dr. D. Luis Ferragut Canals del Departamento de Matemática Aplicada de la Universidad de Salamanca certifica que el trabajo titulado "Métodos de descomposición de dominio con adaptación de mallado en problemas de Convección-Reacción-Difusión" ha sido realizado por Doña Maria Manuela André Alves Simões bajo su dirección en el Departamento de Matemática Aplicada de la Universidad de Salamanca y, que a su entender, este trabajo reúne los requisitos indispensables para que el autor pueda optar al grado de Doctor en Matemáticas por la Universidad de Salamanca.

Y para que así conste, firma el presente certificado en Salamanca a 30 de Septiembre de 2010.

Dr. D. Luis Ferragut Canals

A mis padres y mi hermano.

Agradecimientos

Al terminar este trabajo quiero agradecer a todos los que de alguna forma contribuyeron directa o indirectamente en su realización.

Quiero dar las gracias a mi director de tesis, Luis Ferragut, por toda la sabiduría que me ha transmitido, por su constante disponibilidad, su paciencia y amistad. Gracias Luis, sin tu apoyo y orientación este trabajo no hubiese podido realizarse.

Quiero dar las gracias a Mabel Asensio por su disponibilidad, en particular por su valiosa ayuda en la revisión de este trabajo, por su cariño y amistad.

Quiero dar un agradecimiento especial a José Manuel Cascón por su interés, disponibilidad, por sus sugerencias y sabios consejos. A Santiago Monedero por ofrecerme una visión más física de los problemas del fuego, por su amistad y optimismo. A Francisco Pérez, por su trabajo que me sirvió de inspiración.

Quiero agradecer a los profesores y a los compañeros de la sala de doctorandos del Departamento de Matemáticas de la Universidad de Salamanca, la simpatía y el cariño con que siempre me trataron. En particular a Antonio Almorox, Daniel Sadornil, Ettore Minguzzi y Camelia Trandafir con quienes compartí muchas comidas y discusiones lingüísticas. Gracias por vuestros aportes diarios a mi castellano.

No puedo olvidarme de agradecer a Cristina Malheiro y César Rodrigo que me acogieron en su casa como si fuera una más de la familia, de Pablo M. Chacón y Cecilia Tosar, que con su cariño y amistad nunca dejaron que los fines de semana se me hicieran largos y solitarios.

A mis amigos psicólogos, Diana, Andrés, Noelia y Paula por todas las risas, pinchos, cañas y cenas que habéis compartido conmigo. Gracias Elena Briones y Andreia Albert por animarme, ayudarme, por compartir vuestro tiempo, vuestras alegrías y preocupaciones.

Agradezco a la Fundação para a Ciência e a Tecnologia por financiar esta investigación.

No puedo olvidarme de mis amigos portugueses que a pesar de la distancia no han dejado de mimarme y animarme. En particular, quiero agradecer a mi amiga de siempre Paulinha Tracana, por su presencia en esta etapa tan importante de mi vida. Contigo los agobios y preocupaciones se tornan livianos.

A ti Pedrito, con quien he elegido estar, te agradezco el creer en mí y en mi trabajo. Tu gran paciencia, apoyo, cariño y compañía han sido fundamentales a la hora de terminarlo.

A toda mi familia. A mis padres, Orquidia y Manuel, y a mi hermano Pedro, por mimarme, confiar en mi, valorar mi esfuerzo y estar siempre dispuestos a ayudarme; a mis abuelos, en especial a mí abuelo Álvaro, por transmitirme sus valores, confiar en mi e interesarse en cada uno de mis pasos.

A todos, bem hajam!

Contenidos

Índice de figuras

Introducción

En este trabajo, se presenta un algoritmo numérico para la resolución de problemas elípticos combinando técnicas de descomposición de dominios con el método de elementos finitos adaptativo. Se presenta también la paralelización de dicho algoritmo para máquinas de memoria compartida.

La descomposición del dominio Ω, donde está definido nuestro problema, en diferentes subdominios $\Omega_i, i = 1, ..., nd$, nos permite considerar en cada subdominio diferentes mallados iniciales y aplicar, de forma independiente, un método de elementos finitos adaptativos (MEFA) regido por un estimador a-posteriori en cada uno de los subdominios. El marco teórico de nuestro método se ha desarrollado sobre un problema modelo estacionario y lineal con condiciones de contorno tipo Dirichlet que se puede escribir de forma general del siguiente modo: *Hallar* u *en* $H := H_0^1(\Omega)$ *tal que* $a(u, v) = (f, v) \quad \forall v \in H$, *donde* $(f, g) = \int_\Omega fgdx$. La forma bilineal real $a(., .)$ es continua y elíptica lo que nos garantiza la existencia y unicidad de la solución, siendo esta solución la que queremos aproximar. Se generaliza este algoritmo para la resolución de problemas no lineales, en particular, para problemas de convección-reacción-difusión.

Con respecto a la *descomposición de dominios*, se observa en los últimos años un creciente interés dentro de la comunidad científica en la aplicación de técnicas de descomposición de dominios a la resolución de problemas de Física e Ingeniería. En consecuencia, se han publicado muchos trabajos científicos con la aplicación de estos métodos a diferentes áreas del conocimiento (ver [5], [7], [8], [14], [15], [26], [52]). Uno de los primeros algoritmos para resolver el laplaciano con métodos de descomposición de dominios es el algoritmo de Schwarz. Los métodos de descomposición de dominios se basan en la partición del dominio original, donde el problema está definido, en varios subdominios más pequeños y más simples. Las técnicas de descomposición de dominios son de gran utilidad cuando se aplican a problemas que presentan distintas características en diferentes partes del dominio original, como por ejemplo la no linealidad.

Se pueden encontrar en los trabajos de J.L. Lions y O. Pironneau distintas técnicas de descomposición de dominios, con y sin solapamiento (ver [37], [38]) así como ejemplos de aplicación de esas mismas técnicas a distintos tipos de problemas (ver [24], [36], [39]).

La *paralelización* de métodos de descomposición de dominios (ver [34], [44], [49]) surge de forma natural, dado que el problema en cada subdominio puede ser tratado de forma independiente. Por ello, estos métodos son muy atractivos para ser implementados en máquinas paralelas. En la actualidad, la necesidad de resolver problemas de grandes dimensiones de forma cada vez más rápida y eficiente surge en distintas áreas, como la biomecánica, meteorología e ingeniería o bien en el desarrollo de redes de computador de alta velocidad y desarrollo tecnológico en la construcción de microprocesadores. Esta conlleva la aparición y al

desarrollo de nuevas arquitecturas con gran poder computacional. Estas nuevas arquitecturas están constituidas por varios procesadores, permitiendo así una computación de alto rendimiento. Así, con la aparición de las máquinas paralelas muchos de los algoritmos numéricos clásicos ([32], [49]) que resuelven problemas en las más variadas áreas han sido adaptados a esta nueva realidad. También ante la imposibilidad de hacer tal adaptación, algunos algoritmos se han vuelto a construir de raíz para máquinas paralelas. Surgen también, nuevas formas de almacenar la información ([27]) de modo que esta sea fácilmente accedida y procesada en este tipo de máquinas. Para una visión más general sobre el estado del arte de la computación paralela aplicada, sugerimos la consulta de [33].

Con respecto a los *Métodos de Elementos Finitos Adaptativos* (MEFA), es bien conocido su gran utilidad en la resolución de problemas que presentan algún tipo de singularidad local como por ejemplo: capas límites, singularidades, choques, etc. Para que la calidad de la solución no se vea afectada en las regiones críticas donde la solución no es tan regular se introducen en esas regiones más grados de libertad.

La *adaptación de mallado* regido por un estimador de error a-posteriori, cuyo cálculo depende de los datos del problema y de la solución discreta, es un instrumento esencial para la resolución numérica eficiente de Ecuaciones en Derivadas Parciales (PDEs). El concepto de estimador a-posterior fué introducido en el año de 1978 por Babuška y Rheinboldt [2, 3] y el tema sigue siendo objeto de investigación, véase por ejemplo los trabajos de Verfürth [53, 54] y de Babušca y T. Strouboulis [4]. También Dörfler en [16, 17] construye un algoritmo adaptativo convergente, para la resolución de la ecuación de Poisson y, más adelante, los tra-

bajos de Morin *et. al* (ver [42, 43]) utilizan la idea de Dörfler para desarrolar un algoritmo adaptativo para la formulación clásica del problema elíptico $Au = f$.

Como fuente de inspiración de nuestro algoritmo están los trabajos de Bänsch *et al.* [6] que desarrollan un método adaptativo convergente tipo Uzawa para el problema de Stokes y de F. Andrés [1] que desarrolla métodos adaptativos para problemas no lineales asociados a operadores multívocos.

Nuestro objetivo es construir un algoritmo adaptativo convergente tipo Uzawa (método adaptativo de Uzawa modificado) usando técnicas de descomposición de dominios, al que llamamos AMUADD, para resolver problemas elípticos estacionarios y extender su aplicación a problemas de convección-reacción-difusión así como desarrollar una versión paralela del mismo.

Empezamos considerando un problema lineal estacionario definido en un dominio Ω, descomponemos el dominio en dos subdominios Ω_1 y Ω_2, Γ_{12} es la frontera interna entre los dos subdominios, y aplicamos en cada subdominio un método de elementos finitos adaptativo usando un refinamiento basado en un estimador de error a-posteriori.

El punto de partida es la **Formulación híbrida Primal** de un problema elíptico: Encontrar $(u, p) \in \mathbb{V} \times \Lambda$ tal que:

$$\begin{aligned} \sum_i \int_{\Omega_i} \mathbf{D}\nabla u \nabla v + \sum_i \int_{\partial\Omega_i} pv &= \sum_i \int_{\Omega_i} fv \quad && \forall v \in \mathbb{V} \\ \int_{\Gamma_{12}} [u]\mu &= 0 && \forall \mu \in \mathbb{M} \end{aligned}$$

con $i = 1, 2$ y $[u] = u_1 - u_2$.

Es bien conocido que la rapidez de convergencia del algoritmo de Uzawa es

muy baja en la resolución de este tipo de problemas. En este trabajo modificamos el algoritmo de Uzawa de dos formas: La primera modificación consiste en el uso de diferentes operadores auxiliares para resolver la ecuación $\int_{\Gamma_{12}}[u]\mu = 0$, con el fin de acelerar la convergencia. La segunda, consiste en la introducción de adaptación en las mallas (Algoritmo adaptativo de Uzawa modificado). A continuación se describe el algoritmo:

Escoger parámetros $\rho > 0$ tal que $\sigma := \|I - \rho S\|_{\mathcal{L}(\Lambda,\Lambda)} < 1$, $0 < \gamma < 1$ y $\varepsilon_0 > 0$; tomar j=1.

1. Dado un espacio de elementos finitos $\mathbb{V}_0$ y una aproximación inicial $P_0 \in \mathbb{M}_0$.
2. Calcula U_j en $\mathbb{V}_j$.
3. Actualiza P_j en $\mathbb{M}_j$ usando P_{j-1} y U_j.
4. $\varepsilon_j \leftarrow \gamma\varepsilon_{j-1}$.
5. Calcula T_j por adaptación de la malla T_{j-1}, tal que $|U_j - u| < \varepsilon_j|$
6. $j \leftarrow j + 1$.
7. Ir para el paso 2.

Donde S es el complemento de Schur asociado al algoritmo de Uzawa y el par (U_j, P_j) es la solución discreta para el problema aproximado.

Demostramos la convergencia del método con respecto a la solución discreta en el espacio correspondiente a un mallado uniforme suficientemente fino. Obteremos el siguiente resultado de convergencia:

Sea (U_j, P_j) la sucesión de soluciones obtenida con el algoritmo adaptativo de Uzawa modificado. Existen constantes positivas C y $\delta < 1$ tales que:

$$\|u - U_j\|_{\mathbb{V}} + \|p - P_j\|_{\mathbb{M}} \leq C\delta^j$$

donde $\mathbb{V}$ y $\mathbb{M}$ son espacios funcionales adecuados; u y p son las soluciones discretas en un mallado suficientemente fino. Es decir, dada una tolerancia $\varepsilon > 0$ y

partiendo de una malla inicial grosera en cada uno de los subdominios considerados, el algoritmo debe garantizar que los errores obtenidos en cada uno de los subdominios estén por debajo de esa tolerancia después de un número finito de pasos.

La descomposición del dominio original Ω en varios subdominios Ω_i nos permite considerar diferentes mallados iniciales y hacer adaptación de mallados de forma independiente en cada uno de ellos. En cada subdominio se aplica un método de elementos finitos adaptativo (MEFA) regido por un estimador de error a-posteriori η, del tipo,

$$|u - U_H| \leq \eta = (\sum_{T \in \mathcal{T}_H} \eta_T^2)^{\frac{1}{2}}$$

Así, dada una tolerancia prefijada tol, mientras

$$\eta > (tol * \frac{area\Omega_i}{area\Omega})$$

se refina la malla del subdominio Ω_i, en concreto, se refinan los elementos de la triangulación que presenten mayor error. La razón por la que en la condición anterior se relaciona la tolerancia tol con la área relativa de cada subdominio Ω_i es porque, en la práctica, con este criterio se generan mallas con menor número de grados de libertad, sin prejuicio en los errores presentados por la solución aproximada obtenida. Es decir, cuando utilizamos dicha condición en substitución de la condición que se utilizaría de forma natural en este tipo de problemas, $\eta > tol$, el esfuerzo computacional realizado en el cálculo de la solución aproximada disminuye considerablemente.

En la literatura, existen varias técnicas para obtener el estimador a-posteriori ([54]), en nuestro caso, hemos optado por acotar el error con base en el cálculo del residuo de la solución discreta U_H en una malla dada $\mathcal{T}_H$ y para marcar los

elementos a refinar utilizamos la estrategia de marcado del máximo (se marcan para refinar los elementos que presentan un indicador de error más alto). Dado un parámetro $\gamma \in (0,1)$ se marcan para refinar todos los elementos T de la malla $\mathcal{T}_H$ que verifican,

$$\eta_T > \gamma \max_{T' \in \mathcal{T}_H} \eta_{T'}.$$

Como se sabe, valores pequeños de γ generan mallas muy finas, mientras que valores de γ próximos de 1, generan muchas iteraciones antes de alcanzar la tolerancia prefijada. Generalmente se suele tomar $\gamma \approx 0.5$ ([53]).

Mostramos algunos resultados numéricos obtenidos por aplicación del algoritmo AMUADD, en particular a la resolución de problemas con alguna singularidad en una región bien definida en el dominio original. De este modo, tenemos la posibilidad de hacer una descomposición de dominio aislando la singularidad en un subdominio dado. Las experiencias numéricas muestran que el método se puede adaptar tanto a problemas no lineales como a problemas de convección--reacción-difusión. En cualquiera de los casos el tiempo de cálculo y los recursos computacionales utilizados son menores que sin la adaptación de mallados en los subdominios. Mostramos también como el AMUADD es apropiado para ser usado en máquinas paralelas.

Las experiencias numéricas han sido programadas mediante la librería de elementos finitos neptuno++, neptunoDD++ y freefem++. Los cálculos se han realizado sobre una estación de trabajo DELL PRECISION^{TM} 690 equipada con 2 procesadores de 64-bits Dual-Core Intel® Xeon® serie 5000, un total de cuatro núcleos de ejecución a 3.0GHz, 1333 FSB, 4MB L2 Cache, 80 watts.

El trabajo está organizado según se describe a continuación.

En el capítulo §1 se fija la notación y tipografía utilizada en el trabajo. Se introducen también algunos conceptos básicos de análisis funcional de los que haremos uso.

En el capítulo §2 se considera el problema abstracto, su formulación variacional orientada a la descomposición de dominios. Se describe el algoritmo AMUADD que resulta de la combinación de una iteración de Uzawa y un método de elementos finitos adaptativo y convergente para problemas elípticos con técnicas de descomposición de dominio. Se obtiene un estimador a-posteriori basado en la estimación del residuo y se demuestra la convergencia del algoritmo.

En el capítulo §3 se aplica nuestro algoritmo a un problema estacionario con una singularidad en una región bien definida del dominio, considerando diferentes descomposiciones del dominio original y se presentan los resultados.

En el capítulo §4 se comentan algunos detalles que nos parecen más relevantes en la programación de nuestro algoritmo así como la descripción de algunas de las funciones programadas por la autora.

En el capítulo §5 se extiende la aplicación del AMUADD a un problema de convección-reacción-difusión, se analizan los resultados obtenidos y se incluye el correspondiente código del programa.

El capítulo §6 está dedicado al desarrollo y a la implementación paralela del AMUADD; se incluye parte del software desarrollado.

Para finalizar, en el capítulo §7 presentamos algunas conclusiones y algunos de los problemas abiertos que nos parecen más interesantes.

Las principales aportaciones de esta tesis se incluyen a continuación:

- Formulación de un algoritmo de resolución AMUADD y resultados de convergencia.
- Adaptación de la estimación a-posteriori para la descomposición de dominios.
- Implementación de AMUADD para problemas de convección-reacción-difusión.
- Implementación de AMUADD en máquinas paralelas.
- Aportación de software al neptuno++ con un conjunto de rutinas para descomposición de dominios que llamamos neptunoDD++.

Capítulo 1

Preliminares

En este capítulo fijaremos la notación y tipografía utilizada en el trabajo. Introduciremos conceptos básicos de análisis funcional (espacios funcionales, operadores, normas, ...) necesarios para facilitar la comprensión.

1.1. Generalidades

Enumeramos a continuación las reglas tipográficas que se tendrán en cuenta:

- Las ecuaciones o expresiones matemáticas se referencian con un número entre paréntesis.

- Los problemas, condiciones, definiciones, algoritmos, lemas y teoremas se referencian con el nombre completo o sólo la inicial en mayúsculas seguida de un número.
  (Ejemplo: Problema 1.1 o P.1.1, Condición 2.1 o C.2.1).

- En las referencias a los capítulos, secciones y subsecciones se antepone el símbolo § a la numeración correspondiente.

- Las funciones que pertenezcan a los espacios variacionales, asociados a las formulaciones débiles de las respectivas EDPs, se representarán con letras minúsculas. En cambio, las funciones de los espacios discretos utilizados para la aproximación numérica se escribirán con letras mayúsculas.

- Si X es un espacio normado,

  - $\|\cdot\|_X$ denota la norma de X.
  - $(\cdot,\cdot)$ simboliza el producto escalar.
  - $\langle\cdot,\cdot\rangle$ representa el producto de dualidad.
  - X^* indica el espacio dual de X.
  - $\|\cdot\|_{X^*}$ representa la norma dual, es decir,

$$\|w\|_{X^*} := \sup_{\substack{x\in X\\ x\neq 0}} \frac{\langle w, x\rangle}{\|x\|_X}$$

  - Si $(Y, \|\cdot\|_Y)$ es otro espacio normado, y $T : X \to Y$ es una aplicación lineal se define,

$$\|T\|_{\mathcal{L}(X,Y)} := \sup_{\substack{x\in X\\ x\neq 0}} \frac{\|T(x)\|_Y}{\|x\|_X}$$

1.2. Espacios Funcionales

Sea $\Omega \subset \mathbb{R}^d$ un abierto con frontera poligonal $\partial\Omega := \Gamma$ y sea $p \in \mathbb{R}$ con $1 \leq p < \infty$. Para cada abierto Ω,

$$L^p(\Omega) := \{f : \Omega \to \mathbb{R}; f \text{ medible}, \int_\Omega |f|^p < \infty\}$$

Si $p = \infty$,

$$L^\infty(\Omega) := \{f : \Omega \to \mathbb{R}; f \text{ medible}, \sup_\Omega |f| < \infty\}$$

El espacio de Hilbert de funciones escalares de orden m sobre Ω se define,

$$\begin{aligned} &H^0(\Omega) := L^0(\Omega), && m = 0 \\ &H^m(\Omega) := \{v \in L^2(\Omega) :\ \partial^\alpha v \in L^2(\Omega); \forall \alpha \in \mathbb{N}^d\, |\alpha| \leq m, \}, && m \geq 1 \end{aligned}$$

dotados de las siguientes normas y seminormas,

$$\begin{aligned} \|v\|_{0,\Omega} &:= \Big(\int_\Omega v^2 \Big)^{1/2}, \\ \|v\|_{m,\Omega} &:= \Big(\sum_{\alpha \leq m} \int_\Omega |\partial^\alpha v|^2 dx \Big)^{1/2}, \quad m \geq 1 \\ |v|_{m,\Omega} &:= \Big(\sum_{\alpha = m} \int_\Omega |\partial^\alpha v|^2 dx \Big)^{1/2}, \quad m \geq 1 \end{aligned}$$

Se definen los siguientes operadores diferenciales,

$$\nabla v := \begin{pmatrix} \frac{\partial v}{\partial x_1} \\ \frac{\partial v}{\partial x_2} \end{pmatrix}, \qquad \Delta v := \frac{\partial^2 v}{\partial x_1^2} + \frac{\partial^2 v}{\partial x_2^2}$$

Si u y v son funciones suficientemente regulares, del *teorema de Stokes* en el plano se deduce la fórmula de integración por partes,

$$(\Delta u, v) = -(\nabla u, \nabla v) + \langle \nabla u \cdot \boldsymbol{n}, v \rangle_{\partial\Omega}, \quad \forall u, v \in H^1(\Omega),$$

donde $\boldsymbol{n}$ denota el vector normal unitario exterior en la frontera $\partial\Omega$ y $\langle \cdot, \cdot \rangle_{\partial\Omega}$ es el producto de dualidad entre $H^{-\frac{1}{2}}(\partial\Omega)$ y $H^{\frac{1}{2}}(\partial\Omega)$.

En las EDPs las condiciones de frontera serán de tipo Dirichlet/Neumman. En ocasiones, las condiciones de frontera se incluyen directamente en las definiciones de los espacios variacionales. Así introducimos,

$$H_0^1(\Omega) := \{v \in H^1(\Omega) :\ v|_\Gamma = 0\},$$

dotado de la norma $|\cdot|_{1,\Omega}$.

Para la descomposición de dominios consideramos que el dominio inicial Ω está descompuesto en subdominios $\Omega_i, i = 1, ..., nd$, donde nd es el número de subdominios y $\Gamma_i = \partial\Omega_i \cap \partial\Omega$. Se definen nuevos espacios,

$$\mathbb{V} := \prod_{i=1,...nd} H^1_{0,\Gamma_i}(\Omega_i)$$

donde

$$H^1_{0,\Gamma_i}(\Omega_i) = \left\{v \in H^1(\Omega_i) : v|_{\Gamma_i} = 0\right\}$$

dotado de la norma $\|v\|_{\mathbb{V}} = (\sum_{i=1}^{nd} |v|^2_{1,\Omega_i})^{1/2}$ y el espacio

$$\Lambda = \prod_{\substack{i,j=1,...nd \\ i\neq j}} H^{-\frac{1}{2}}(\Gamma_{ij})$$

donde $H^{-\frac{1}{2}}(\Gamma_{ij})$ es el espacio dual del espacio de las trazas $H^{\frac{1}{2}}(\Gamma_{ij})$, constituido por funciones que están definidas en la frontera interna, $\Gamma_{ij} = \partial\Omega_i \cap \partial\Omega_j$, para $i \neq j$ dado por,

$$\begin{aligned} H^{\frac{1}{2}}(\Gamma_{ij}) &= \left\{\mu : \mu = v|_{\Gamma_{ij}}, \quad v \in H^1_{0,\Gamma_i}(\Omega_i)\right\} \\ &= \left\{\mu : \mu = v|_{\Gamma_{ij}}, \quad v \in H^1_{0,\Gamma_j}(\Omega_j)\right\} \end{aligned}$$

1.3. Espacios Discretos

Sea $\mathcal{T}_h := \{T\}$ una triangulación regular conforme de Ω (ver [12]). Dado un elemento $T \in \mathcal{T}_h$, h_T denotará su diámetro. Llamaremos $\mathcal{S}_h$ al conjunto de aristas $\mathcal{S}$ de $\mathcal{T}_h$; $h_{\mathcal{S}}$ será la longitud de la arista $\mathcal{S}$.

Sea $\mathcal{T}_{h1} := \{T'\}$ una partición de la frontera interna Γ_{ij} entre dos subdominios Ω_i e Ω_j en subintervalos. Dado un elemento $T' \in \mathcal{T}_{h1}$, h_{1T} denotará su diámetro.

Durante el proceso adaptativo la malla $\mathcal{T}_h$ se modificará refinando algunos de sus elementos. En general, en este trabajo, el subíndice h se suprimirá o será sustituido por un índice numérico que hará alusión a la iteración del proceso adaptativo.

Para aproximar la solución y el multiplicador de Lagrange que intervienen en la formulación de los problemas resueltos es necesario construir ciertos subespacios discretos de los espacios $H_0^1(\Omega)$ y $L^2(\Gamma_{lk})$ sobre las triangulaciones $\mathcal{T}_h$ y $\mathcal{T}_{h1}$, respectivamente. En este trabajo se utilizan los pares de espacios discretos $(\mathcal{P}^1(\mathcal{T}_h), \mathcal{P}^0(\mathcal{T}_{h1}))$ y $(\mathcal{P}^1(\mathcal{T}_h), \mathcal{P}^1(\mathcal{T}_{h1}))$ que se definen a continuación:

- $\mathcal{P}^0(\mathcal{T}_{h1})$ son los elementos de Lagrange de grado 0 discontinuos, es decir, el espacio de funciones que restringidas a cada elemento $T' \in \mathcal{T}_{h1}$ son funciones constantes:

$$\mathcal{P}^0(\mathcal{T}_{h1}) := \{v \in L^2(\Gamma_{lk}) : \ v_{|T'} \in P^0(T') \quad \forall\, T' \in \mathcal{T}_{h1}\}$$

- $\mathcal{P}^1(\mathcal{T}_{h1})$ son los elementos de Lagrange de grado 1 continuos, es decir, el espacio de funciones que restringidas a cada elemento $T' \in \mathcal{T}_{h1}$ son polinomios de grado 1:

$$\mathcal{P}^1(\mathcal{T}_{h1}) := \{v \in H^1(\Gamma_{lk}) : \ v_{|T'} \in P^1(T') \quad \forall\, T' \in \mathcal{T}_{h1}\}$$

- $\mathcal{P}^1(\mathcal{T}_h)$ son los elementos de Lagrange de grado 1 continuos, es decir, polinomios de grado 1 y que se anulan en la frontera $\partial\Omega$:

$$\mathcal{P}^1(\mathcal{T}_h) := \{v \in \mathcal{C}(\overline{\Omega}) : \ v_{|T} \in P^1(T) \quad \forall\, T \in \mathcal{T}_h\} \cap H_0^1(\Omega)$$

Capítulo 2

Problema General Abstracto

El objetivo de este capítulo es introducir nuestro algoritmo adaptativo para un problema general abstracto. Daremos el marco funcional, los espacios y las hipótesis necesarias para su aplicación.

2.1. Formulación Variacional

En esta sección planteamos el problema modelo en términos de su formulación variacional orientada a la descomposición de dominios. Sin pérdida de generalidad vamos a considerar la descomposición del dominio original Ω en dos subdominios Ω_1 e Ω_2.

Sean $(H, \| \quad \|)$ y $(\mathbb{M}, \| \quad \|_{\mathbb{M}})$ espacios de Hilbert con sus respectivas normas; $H := H_0^1(\Omega)$ es el espacio de funciones de $L^2(\Omega)$ en que las primeras derivadas en el sentido de las distribuciones están en $L^2(\Omega)$ y se anulan en la frontera $\partial\Omega$; $\mathbb{M} := L^2(\Gamma_{12})$, son las funciones de cuadrado integrable definidas en la frontera Γ_{12} ; H^* y $\mathbb{M}^*$ son los correspondientes espacios duales. Habitualmente identifi-

camos $\mathbb{M}$ y $\mathbb{M}^*$. Denotamos por $(\cdot,\cdot)$ el producto escalar en $\mathbb{L}^2$ y $\langle\cdot,\cdot\rangle$ el producto de dualidad entre H^* y H,

$$\begin{array}{rccccl} \langle\cdot,\cdot\rangle: & H^* & \times & H & \to \mathbb{R} \\ & f & , & x & \to \langle f, x\rangle \end{array}$$

Sea $a(\cdot,\cdot)$ una forma bilineal real y continua en H,

$$\begin{array}{rccccl} a: & H & \times & H & \to \mathbb{R} \\ & u & , & v & \to a(u,v) \end{array}$$

definida por

$$a(u,v) = \sum \int_\Omega d_{kl} \frac{\partial u}{\partial x_l} \frac{\partial v}{\partial x_k} dx, \quad \forall u, v \in H$$

donde $D = (d_{kl})$ es un tensor verificando y $\alpha > 0$

$$d_{kl} \in L^\infty(\Omega), \qquad \sum_{k,l} d_{kl}\xi_k\xi_l \geqslant \alpha \sum_k \xi_k^2, \text{ en } \Omega.$$

y $\alpha > 0$, por tanto, $a(\cdot,\cdot)$ es elíptica en H, esto es,

$$a(v,v) \geq \alpha \|v\|_H^2, \qquad \forall v \in H$$

y a partir de ésta se define el operador lineal y continuo

$$A : H \to H^*$$

que viene dado por

$$\langle Au, v\rangle = a(u,v)$$

donde H^* es el espacio dual de H. Por tanto,

Definición 2.1. $a(\cdot,\cdot)$ define una norma en H equivalente a la norma que está definida en H,

$$\|v\|_A := a(v,v)^{1/2} \tag{2.1}$$

Esta equivalencia de normas resulta de: Si Ω es acotado, la seminorma $|v|_{1,\Omega}$ es una norma sobre $H_0^1(\Omega)$ equivalente a la norma inducida $H^1(\Omega)$, esto es,

$$\exists\ C_1, C_2 > 0 \text{ tales que } C_1\|v\|_{1,\Omega} \leq |v|_{1,\Omega} \leq C_2\|v\|_{1,\Omega} \quad \forall\ v \in H_0^1(\Omega) \tag{2.2}$$

Definición 2.2. Se define la norma de la aplicación bilineal $a(\cdot,\cdot)$,

$$\|a\| := \sup_{\substack{u,v\in H\\ u,v\neq 0}} \frac{a(u,v)}{\|u\|\|v\|} \tag{2.3}$$

Dado $f \in L^2(\Omega)$ el problema modelo estacionario se enuncia

Problema 2.1. Hallar u en H tal que

$$a(u,v) = (f,v) \quad \forall v \in H \tag{2.4}$$

donde $(f,g) = \int_\Omega fg dx$.

Teorema 2.1. *Si $a(\cdot,\cdot)$ y f satisfacen las condiciones reseñadas anteriormente, el Problema 2.1 tiene solución única.*

Demostración. Ver [46]. □

2.2. Problema Modelo con Descomposición de Dominios

Vamos a empezar por reformular el problema general P.2.1 que queremos resolver con descomposición de dominios. Consideremos que el dominio Ω, abierto, acotado, con frontera regular está constituido por distintos abiertos $\Omega_i, i = 1, ..., nd$ y denotemos por Γ_{ij} la frontera común entre Ω_i y Ω_j. Definimos Γ_i como la intersección de $\partial\Omega$ con $\partial\Omega_i$,

$$\begin{aligned} \bar{\Omega} &= \cup\bar{\Omega}_i & \cap\,\Omega_i &= \emptyset \\ \Gamma_{ij} &= \partial\Omega_i \cap \partial\Omega_j & \Gamma_i &= \partial\Omega \cap \partial\Omega_i \end{aligned}$$

es decir, vamos a considerar una descomposición del dominio Ω sin solapamiento. A continuación y sin pérdida de generalidad, consideraremos la descomposición de Ω en dos subdominios, Ω_1 y Ω_2. Es decir, vamos a considerar que el dominio Ω está constituido por dos partes distintas Ω_1, Ω_2 (ver Figura 2.1), denotemos por Γ_{12} la frontera común entre Ω_1 y Ω_2. Y por Γ_i la intersección de $\partial\Omega$ con $\partial\Omega_i$,

$$\bar{\Omega} = \bar{\Omega}_1 \cup \bar{\Omega}_2 \qquad \Omega_1 \cap \Omega_2 = \emptyset \qquad \Gamma_{12} = \partial\Omega_1 \cap \partial\Omega_2 \qquad \Gamma_i = \partial\Omega \cap \partial\Omega_i \quad i = 1, 2$$

Resolver el problema modelo P.2.1 en $H_0^1(\Omega)$ es equivalente a determinar el par $\{u_1, u_2\}$ tal que

$$u_i \in H^1(\Omega_i) \qquad i = 1, 2 \tag{2.5}$$

$$u_1 = u_2 \qquad \text{en} \quad \Gamma_{12} \tag{2.6}$$

$$u_i = 0 \qquad \text{en} \quad \partial\Omega \cap \partial\Omega_i \tag{2.7}$$

$$(D\nabla u_1).\boldsymbol{n_1} + (D\nabla u_2).\boldsymbol{n_2} = 0 \qquad \text{en} \quad \Gamma_{12} \tag{2.8}$$

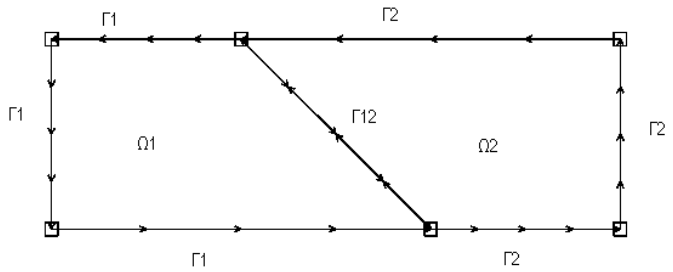


Figura 2.1: Descomposición del dominio Ω

donde en (2.8) $\boldsymbol{n_i}$ denota el vector unitario exterior, normal a la frontera $\partial\Omega_i$ (en Γ_{12} tenemos que $\boldsymbol{n_1} + \boldsymbol{n_2} = 0$). Las condiciones (2.6) y (2.8) imponen respectivamente la continuidad de la solución y del flujo en la frontera interna Γ_{12}.

Vamos a introducir algunas notaciones que usaremos a continuación. Definimos los espacios

$$H^1_{0,\Gamma_i}(\Omega_i) = \left\{ v \in H^1(\Omega_i) : v|_{\Gamma_i} = 0 \right\} \quad para \quad i = 1, 2$$

$$\mathbb{V} := \prod_{i=1,2} H^1_{0,\Gamma_i}(\Omega_i)$$

y

$$\Lambda = H^{-\frac{1}{2}}(\Gamma_{12})$$

siendo $H^{-\frac{1}{2}}(\Gamma_{12})$ el espacio dual del espacio de las trazas $H^{\frac{1}{2}}(\Gamma_{12})$ definido por

$$\begin{aligned} H^{\frac{1}{2}}(\Gamma_{12}) &= \left\{ \mu : \mu = v|_{\Gamma_{12}}, \quad v \in H^1_{0,\Gamma_1}(\Omega_1) \right\} \\ &= \left\{ \mu : \mu = v|_{\Gamma_{12}}, \quad v \in H^1_{0,\Gamma_2}(\Omega_2) \right\} \end{aligned}$$

Otra formulación del problema modelo equivalente a la anterior (2.5)-(2.8) está dada por la formulación Híbrida Primal,

Problema 2.2. Hallar (u,p) $\in \mathbb{V} \times \Lambda$ tales que

$$\sum_l \int_{\Omega_l} D\nabla u \nabla v + \sum_l \int_{\partial\Omega_l} pv = \sum_l \int_{\Omega_l} fv \quad \forall v \in \mathbb{V} \tag{2.9}$$

$$\int_{\Gamma_{12}} [\![u]\!]\mu = 0 \qquad \forall \mu \in \Lambda \tag{2.10}$$

con $k(x) \in L^\infty(\Omega)$ y $[\![u]\!] = u_1 - u_2$.

Dado p, los problemas (2.9) en la incógnita u_i están desacoplados. Para resolver el Problema 2.2 podemos usar el bien conocido Algoritmo de Uzawa (c.f. [25, 23]), que aplicado a nuestro caso particular y, teniendo en cuenta la descomposición de dominios, se escribe:

Algoritmo 2.1. *Dado $p^0 \in \Lambda$ arbitrario y obtenido $p^n \in \Lambda$ se halla u^n resolviendo*

$$\sum_l \int_{\Omega_l} D\nabla u^n \nabla v = \sum_l \int_{\Omega_l} fv - \sum_l \int_{\partial\Omega_l} p^n v \tag{2.11}$$

y se actualiza el multiplicador de Lagrange p^n, utilizando la expresión siguiente para obtener p^{n+1},

$$\int_{\Gamma_{12}} (p^{n+1})'\mu' + \int_{\Gamma_{12}} p^{n+1}\mu = \int_{\Gamma_{12}} (p^n)'\mu' + \int_{\Gamma_{12}} p^n\mu - \rho \int_{\Gamma_{12}} [u^n]v \tag{2.12}$$

o bien, usando una de las siguientes expresiones modificadas:

1.

$$\int_{\Gamma_{12}} p^{n+1}\mu = \int_{\Gamma_{12}} p^n\mu - \rho \int_{\Gamma_{12}} [u^n]v \tag{2.13}$$

2.

$$\int_{\Gamma_{12}} \mathbf{D^{-1}} p^{n+1}\mu = \int_{\Gamma_{12}} \mathbf{D^{-1}} p^n\mu - \rho \int_{\Gamma_{12}} [u^n]v \tag{2.14}$$

En este trabajo se elige la fórmula (2.12) porque ésta nos conduce a mejores resultados al involucrar las derivadas del multiplicador p.

2.3. Uzawa como Algoritmo de Punto Fijo

El Algoritmo 2.1 se puede reformular como *algoritmo de punto fijo*, formulación que, desde el punto de vista del análisis numérico, resulta más útil a nuestros propósitos. La formulación mediante operadores permite incorporar de modo más sencillo la adaptación del mallado a cada subdominio. Así considerando una formulación en términos de operadores y considerando una descomposición del dominio Ω en 2 subdominios tendremos,

$$A_1 u_1 + \gamma_1^* p = f_1 \quad \text{en } H^1_{0,\Gamma_1}(\Omega_1) \tag{2.15}$$

$$A_2 u_2 - \gamma_2^* p = f_2 \quad \text{en } H^1_{0,\Gamma_2}(\Omega_2) \tag{2.16}$$

$$Bu = 0 \quad \text{sobre } \Lambda \tag{2.17}$$

donde γ_i es una aplicación lineal y continua definida por

$$\begin{aligned} \gamma_i : \quad H^1(\Omega_i) &\to L^2(\Gamma_{12}) \\ u_i &\to \gamma_i u_i = u_i|_{\Gamma_{12}} \end{aligned}$$

y γ_1^*, γ_2^* son las correspondientes aplicaciones adjuntas de γ_1 y γ_2 que verifican

$$\langle \gamma^* \mu, v \rangle = {}_{H^{-\frac{1}{2}}}\langle \mu, \gamma v_i \rangle_{H^{\frac{1}{2}}} \qquad \forall v_i \in H^1_{0,\Gamma i}(\Omega_i)$$

Así, con la notación siguiente

$$A = \begin{pmatrix} A_1 & 0 \\ 0 & A_2 \end{pmatrix},\ B = (\gamma_1, -\gamma_2),\ B^* = \begin{pmatrix} \gamma_1^* \\ -\gamma_2^* \end{pmatrix},\ u = \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} \text{ y } f = \begin{pmatrix} f_1 \\ f_2 \end{pmatrix}$$

la ecuación (2.17) se puede escribir en la forma,

$$Bu = 0 <=> (\gamma_1, -\gamma_2) \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} = \gamma_1 u_1 - \gamma_2 u_2 = 0$$

Es decir, el problema (2.15)-(2.17) es equivalente a (2.18)-(2.19) que como vamos a ver a continuación se puede escribir en una sola iteración recursiva de **punto fijo** en la incógnita p.

$$\begin{pmatrix} A_1 & 0 \\ 0 & A_2 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} + \begin{pmatrix} \gamma_1^* \\ -\gamma_2^* \end{pmatrix} p = \begin{pmatrix} f_1 \\ f_2 \end{pmatrix} \tag{2.18}$$

$$(\gamma_1, -\gamma_2) \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} = 0 \tag{2.19}$$

En efecto, si eliminamos u en (2.18) tendremos,

$$\begin{pmatrix} u_1 \\ u_2 \end{pmatrix} = \begin{pmatrix} A_1^{-1} & 0 \\ 0 & A_2^{-1} \end{pmatrix} \begin{pmatrix} f_1 \\ f_2 \end{pmatrix} - \begin{pmatrix} A_1^{-1} & 0 \\ 0 & A_2^{-1} \end{pmatrix} \begin{pmatrix} \gamma_1^* \\ -\gamma_2^* \end{pmatrix} p \tag{2.20}$$

Consideremos aún la siguiente relación en p:

$$Ip = Ip + \rho Bu \quad \text{para } \rho > 0 \tag{2.21}$$

Sustituyendo, ahora, B y u en (2.21) tendremos

$$\begin{aligned} Ip =& Ip + \rho(\gamma_1, -\gamma_2) \begin{pmatrix} A_1^{-1} & 0 \\ 0 & A_2^{-1} \end{pmatrix} \begin{pmatrix} f_1 \\ f_2 \end{pmatrix} \\ & - (\gamma_1, -\gamma_2) \begin{pmatrix} A_1^{-1} & 0 \\ 0 & A_2^{-1} \end{pmatrix} \begin{pmatrix} \gamma_1^* \\ -\gamma_2^* \end{pmatrix} p \end{aligned} \tag{2.22}$$

Pasamos ahora a definir el operador S, **complemento de Schur** [27].

Definición 2.3. $S : \Lambda \to \Lambda$

$$S := BA^{-1}B^*$$

En la ecuación (2.22) tendremos en Λ la siguiente iteración de punto fijo

$$p = (I - \rho S)p + \rho BA^{-1}f \tag{2.23}$$

dando esta relación origen al algoritmo adaptativo que vamos aplicar a cada subdominio Ω_1, Ω_2 para el cálculo de la solución del problema estacionario.

Algoritmo 2.2. *Dado p_0, obtenido $p_{j-1} \in \Lambda$, se calcula p_j para $j \geq 1$*

$$p_j = (I - \rho S)p_{j-1} + \rho B A^{-1} f \tag{2.24}$$

En la práctica aplicamos el algoritmo en subespacios de dimensión finita $\mathbb{V}$ y Λ. En ese caso S es simétrica y definida positiva y este algoritmo será convergente eligiendo $\rho > 0$ tal que el operador S verifique

$$\sigma := \|I - \rho S\|_{\mathcal{L}(\Lambda,\Lambda)} < 1. \tag{2.25}$$

2.4. Algoritmo Uzawa Modificado Adaptativo con Descomposición de Dominios AMUADD

El algoritmo que presentamos está inspirado en el trabajo de E. Bänsch, P. Morin y R. H. Nochetto ([6]) en el que aplican un algoritmo similar para resolver el problema de Stokes y, en los trabajos de W. Döfler ([16, 17]) y de F. Andrés ([1]) en los que se aplican diferentes estrategias de adaptación de mallado a diferentes tipos de problemas. La construcción de nuestro algoritmo descansa en la combinación de una iteración de **Uzawa** y un **Método de Elementos Finitos** (MEF) adaptativo y convergente para problemas elípticos con técnicas de descomposición de dominio. En nuestro algoritmo adaptativo la iteración **Uzawa** juega el papel de iterador externo y resuelve la ecuación (2.10) del Problema 2.2. Mientras que, internamente, se aplicará un método de elementos finitos adaptativo (MEFA) regido por un estimador aposteriori. El MEFA resuelve, en realidad,

una aproximación de la ecuación (2.9) del Problema 2.2. La iteración de Uzawa combinada con MEFA la llamamos algoritmo de Uzawa modificado adaptativo.

En esta sección describiremos nuestro algoritmo. Para ello vamos a definir simultáneamente el problema aproximado y el algoritmo de resolución. Para simplificar notaciones vamos a decir que $\mathcal{T}_j$ es el mallado obtenido por refinamiento del mallado $\mathcal{T}_{j-1}$ y que los correspondientes espacios de funciones de elementos finitos se denotan por $(\mathbb{V}_j, \mathbb{M}_j)$ y $(\mathbb{V}_{j-1}, \mathbb{M}_{j-1})$.

Consideremos el par de sucesiones de espacios de elementos finitos encajados de $\mathbb{V}$ y $\mathbb{M}$:

$$\begin{aligned} \mathbb{V}_0 \subset \mathbb{V}_1 \subset \ldots \subset \mathbb{V}_{j-1} \subset \mathbb{V}_j \subset \ldots \subset \mathbb{V}_J \\ \mathbb{M}_0 \subset \mathbb{M}_1 \subset \ldots \subset \mathbb{M}_{j-1} \subset \mathbb{M}_j \subset \ldots \subset \mathbb{M}_J \end{aligned} \tag{2.26}$$

Consideremos $P_0 \in \mathbb{M}_0$, $\varepsilon_0 > 0$, $0 < \gamma < 1$. Para cada $j = 1, ..., J$ obtenido $P_{j-1} \in \mathbb{M}_{j-1}$, $u_j \in \mathbb{V}_J$ es solución del problema auxiliar,

$$Au_j = f - B^* P_{j-1} \tag{2.27}$$

que se puede escribir en la forma equivalente

$$a(u_j, v) = \langle f, v \rangle - (P_{j-1}, Bv) \quad \forall v \in \mathbb{V}_J \tag{2.28}$$

donde de aquí en adelante $a(.,.)$ denota

$$a(u, v) = \sum_l \int_{\Omega_l} D \nabla u \nabla v \tag{2.29}$$

La ecuación (2.28) se resuelve mediante un MEF adaptativo, es decir, fijada una tolerancia ε_j, hallar una aproximación U_j en $\mathbb{V}_j$ solución de la ecuación discretizada de (2.28), es decir, resolvemos

$$a(U_j, V) = \langle f, V \rangle - (P_{j-1}, BV) \qquad \forall V \in \mathbb{V}_j \tag{2.30}$$

tal que

$$\|u_j - U_j\|_{\mathbb{V}} < \varepsilon_j \tag{2.31}$$

Es decir, construimos $\mathbb{V}_j$ tal que $\mathbb{V}_{j-1} \subset \mathbb{V}_j \subset \mathbb{V}_J$ y $U_j \in \mathbb{V}_j$, solución discreta de (2.28), verificando,

$$\|U_j - u_j\|_{\mathbb{V}} \leq C\varepsilon_j, \tag{2.32}$$

donde C es una constante independiente de j y $\varepsilon_j < \gamma\varepsilon_{j-1}$.
Una vez obtenido U_j actualizamos P_{j-1} en $\mathbb{M}_j$ donde $\mathbb{M}_{j-1} \subset \mathbb{M}_j \subset \mathbb{M}_J$, a través del cálculo,

$$P_j = P_{j-1} + \rho B_j U_j \tag{2.33}$$

siendo $B_j : \mathbb{V}_j \to \mathbb{M}_j$ el operador definido por

$$B_j = B|_{\mathbb{V}_j} \tag{2.34}$$

es decir, la expresión (2.33) se puede volver a escribir de forma equivalente,

$$P_j = P_{j-1} + \rho B U_j \tag{2.35}$$

Con el propósito de clarificar el algoritmo, exponemos brevemente las etapas que realiza en la siguiente tabla. El paso 2 del algoritmo se aplica de forma independiente a cada uno de los subdominios.

Algoritmo Uzawa Modificado Adaptativo con Descomposición de Dominios (AMUADD)

Escoger parámetros $\rho > 0$ verificando (2.25), $0 < \gamma < 1$, $\varepsilon_0 > 0$; Fijar $j = 1$.

1. Dado un espacio de dimensión finita $\mathbb{V}_0$ y aproximación inicial $P_0 \in \mathbb{M}_0$.
2. Calcular U_j en $\mathbb{V}_j$.
3. Actualizar P_j en $\mathbb{M}_j$ usando P_{j-1} y U_j.
4. Obtener $\mathcal{T}_j$ por adaptación del mallado $\mathcal{T}_{j-1}$.
5. $\varepsilon_j \leftarrow \gamma\varepsilon_{j-1}$.
6. $j \leftarrow j + 1$.
7. Ir al paso 2.

2.5. Descripción del algoritmo en cada subdominio

En esta sección describimos con detalle los pasos más importantes del algoritmo. Para resolver el problema discreto empezamos por elegir una triangulación $\mathcal{T}_h$, regular y conforme de Ω_l (ver [12]) y una partición $\mathcal{T}_{h1}$ regular y conforme de la frontera interna Γ_{lk}, con $\Gamma_{lk} = \partial\Omega_l \cap \partial\Omega_k$. La versión discreta del Problema 2.2 se construye a partir de la elección de un par de espacios de elementos finitos donde se aproximan la variable u_l y el multiplicador p. Aproximamos $u_l \in H^1_{0,\Gamma_l}(\Omega_l)$ en el espacio de elementos de Lagrange de grado 1 continuos, funciones lineales en cada elemento T sobre la triangulación $\mathcal{T}_h$ y continuas en Ω_l. La aproximación del multiplicador $p \in H^{-\frac{1}{2}}(\Gamma_{lk})$, la buscamos en el espacio de elementos de Lagrange de grado 0 discontinuos (funciones constantes a trozos sobre la partición $\mathcal{T}_{h1}$) o en el espacio de elementos de Lagrange de grado 1 continuos (funciones lineales

a trozos sobre la partición $\mathcal{T}_{h1}$). Así, definimos los espacios

$$\mathbb{V}_h := \{v_h \in \mathcal{C}(\overline{\Omega}) : \ v_{h|T} \in P^1(T) \quad \forall\, T \in \mathcal{T}_h\} \cap H_0^1(\Omega) \tag{2.36}$$

y

$$\mathbb{M}_h := \{v_h \in H^1(\Gamma_{lk}) : \ v_{h|T} \in P^1(T) \quad \forall\, T \in \mathcal{T}_{h1}\} \tag{2.37}$$

si la adaptación se hace con (2.12), o

$$\mathbb{M}_h := \{v_h \in L^2(\Gamma_{lk}) : \ v_{h|T} \in P^0(T) \quad \forall\, T \in \mathcal{T}_{h1}\} \tag{2.38}$$

si la adaptación se hace con (2.13).

Empezamos considerando una triangulación $\mathcal{T}_0$ en el dominio Ω_i, que corresponde a un mallado grosero. Obtenido P_{j-1} resolvemos

$$a(U_j, V) = \langle f, V\rangle - (P_{j-1}, BV) \qquad \forall V \in \mathbb{V}_j \tag{2.39}$$

mediante un método de Gradiente Conjugado. Calculamos una estimación a-posteriori del error ([2], [53]),

$$\|u_j - U_j\|_{\mathbb{V}} \leq C\eta(U_j) \leq C\Big(\sum_{T\in\mathcal{T}_j} \eta_T^2\Big)^{1/2} \tag{2.40}$$

donde $\eta(U_j)$ es el estimador del error global, y η_T es el indicador del error local definido más adelante en (2.48).

A continuación marcamos los elementos donde el indicador es grande para refinamiento, mientras aquellos elementos con un indicador de error pequeño no se modifican. El refinamiento de los elementos marcados en la malla $\mathcal{T}_h$ se hace usando el algoritmo de refinamiento Rivara $4T$ (ver [47, 48]) y los elementos marcados del mallado $\mathcal{T}_{h1}$ se refinan por división en dos partes iguales. Como estrategia de marcado de los elementos usamos la estrategia del máximo.

Dado un parámetro $\gamma \in (0,1)$ se marcan para refinar todos los elementos $\bar{T}$ de la malla $\mathcal{T}_h$ que verifican

$$\eta_{\bar{T}} > \gamma \operatorname{máx} \eta_T, \quad \forall \quad T \in \mathcal{T}_h$$

Cuando se toman valores de γ pequeños, se obtiene mallas muy refinadas, mientras que para valores de γ próximos a 1 generan muchas iteraciones antes de alcanzar la tolerancia prefijada. Así que, en la práctica se suele considerar $\gamma \approx 0.5$ ([53]). Dada una tolerancia *tol* para el error y mientras se verifique que $\eta > tol$, el algoritmo refina la malla para reducir el error y resuelve de nuevo el problema en la malla fina.

2.6. Estimación a-posteriori

Las estimaciones a-posteriori son cantidades calculables que dependen de los datos del problema, de la solución discreta y que nos dan información de la calidad de la aproximación obtenida. El concepto de estimación *a-posteriori* fue introducido por Babuška y Rheinboldt en los trabajos [3, 2] a finales de los años 70. Más recientemente los trabajos de P. Morin, R. H. Nochetto y K. G. Siebert [42, 43] extienden este concepto y lo aplican a métodos de elementos finitos adaptativos. En esta sección se obtiene un estimador a-posteriori del error, basado en la estimación del residuo (2.42) para el problema (2.30), que es una cota superior para el error. Esta cota superior o cota de fiabilidad, es de carácter global y permite obtener una solución numérica con una precisión por debajo de una tolerancia dada. En este apartado utilizaremos los subíndices H y h para referirnos a los objetos asociados a dos mallas consecutivas (es decir, $\mathcal{T}_h$ se obtiene refinando la triangulación $\mathcal{T}_H$ de Ω). El par (U_H, P_H) se refiere a solución discreta sobre $\mathcal{T}_H$

y $\mathcal{T}_{H1}$ en los espacios $\mathbb{V}_H$ y $\mathbb{M}_H$ respectivamente. Los elementos de la triangulación $\mathcal{T}_H$ se denotarán por T, y $S \in \mathcal{S}_H$ será el conjunto de aristas de $\mathcal{T}_H$ que no pertenecen a ninguna frontera $\Gamma_i = \partial\Omega \cap \partial\Omega_i$, $i = 1, 2$. De igual forma los elementos de la partición $\mathcal{T}_{H1}$ se denotarán por T', y $S' \in \mathcal{S}'_H$ será el conjunto de aristas de $\mathcal{T}_H$ que están en la frontera interna $\Gamma_{12} = \partial\Omega_1 \cap \partial\Omega_2$.

En cada subdominio Ω_l, estimaremos el error $u - U_H$ en la norma definida en $H^1_{0,\Gamma_l}(\Omega_l)$ siguiente

$$|\cdot| : v \in H^1_{0,\Gamma_l}(\Omega_l) \longrightarrow |v| = \Big(\int_{\Omega_l} D\nabla v \nabla v \Big)^{\frac{1}{2}} \tag{2.41}$$

Siguiendo a [54], para estimar $|u - U_H|$ introducimos el residuo $res(U_H)$ asociado a una solución aproximada U_H

$$\langle res(U_H), v \rangle = \int_{\Omega_l} fv - \int_{\Gamma_{lk}} P_H v - \int_{\Omega_l} D\nabla U_H \nabla v \tag{2.42}$$

donde en la expresión anterior los valores de U_H, v y f se entienden como la restricción al correspondiente dominio Ω_l de los valores de estas funciones. Para acotar el lado derecho de (2.42) introducimos un operador de interpolación con propiedades óptimas de aproximación y estabilidad.

Definición 2.4. Sea $C_H : \mathbb{V} \longrightarrow \mathbb{V}_H$ el operador de interpolación de Clément ([13]). C_H satisface las siguientes propiedades,

Invarianza: $C_H V = V \quad \forall V \in \mathbb{V}_H$

Estabilidad: $\|C_H v\|_W \leq c\|v\|_W \quad \forall v \in \mathbb{V} \quad (W = L^2(\Omega), H^1_0(\Omega))$.

Aproximación:

$$\|v - C_H v\|_{0,T} \leq c_1 h_T \|\nabla v\|_{0,\Omega_T} \quad \forall v \in \mathbb{V}, \tag{2.43}$$

$$\|v - C_H v\|_{0,S} \leq c_2 h_S^{1/2} \|\nabla v\|_{0,\Omega_S} \quad \forall v \in \mathbb{V}, \tag{2.44}$$

donde Ω_T y Ω_S designan para cada triángulo $T \in \mathcal{T}_H$ y para cada arista $S \in (\mathcal{S}_H \cup \mathcal{S'}_H)$,

$$\Omega_T = \{T^* \in \mathcal{T}_H : T \text{ y } T^* \text{ tienen un vértice común}\},$$
$$\Omega_S := \{T' \in \mathcal{T}_H : T' \text{ tiene un vértice en común con S}\}$$

y c es una constante que depende del ángulo mínimo de la triangulación de $\mathcal{T}_H$.

Vamos ahora a estimar el valor de $\langle res(U_H), v\rangle$ que nos proporcionará una estimación del error $|u - U_H|$. Para cada $v \in \mathbb{V}$ y cada subdominio Ω_l, integrando por partes, se tiene,

$$\begin{aligned}
\langle res(U_H), v\rangle &= \int_{\Omega_l} fv - \int_{\Gamma_{lk}} P_H v - \int_{\Omega_l} D\nabla U_H \nabla v \\
&= \int_{\Omega_l} fv - \int_{\Gamma_{lk}} P_H v - \sum_{T\in\mathcal{T}_H} \int_T D\nabla U_H \nabla v \\
&= \int_{\Omega_l} fv - \int_{\Gamma_{lk}} P_H v + \sum_{T\in\mathcal{T}_H} \{\int_T \nabla(D\nabla U_H)v - \int_{\partial T} D\nabla U_H.n_T v\} \\
&= \sum_{T\in\mathcal{T}_H} \int_T (f + \nabla(D\nabla U_H))v - \sum_{S'\in\mathcal{S'}_H} \int_{S'} (P_H - D\nabla U_H.n_{S'})v \\
&\quad - \sum_{S\in\mathcal{S}_H} \int_S [\![D\nabla U_H.n_S]\!]_s v \qquad (2.45)
\end{aligned}$$

donde $[\![D\nabla U_H.n_S]\!]$ son los saltos de $D\nabla U_H.n_S$ a través de la arista S. Tomando en (2.45) $w \in \mathbb{V}$, aplicando (2.43-2.44) y la desigualdad de Cauchy-Schwartz,

$$\begin{aligned}
\langle res(U_H), w\rangle = &\sum_{T\in\mathcal{T}_H} \int_T (f + \nabla(D\nabla U_H))(w - C_H w) \\
&- \sum_{S'\in\mathcal{S'}_H} \int_{S'} (P_H - D\nabla U_H.n_{S'})(w - c_H w) \\
&- \sum_{S\in\mathcal{S}_H} \int_S [\![D\nabla U_H.n_S]\!]_s (w - C_H w)
\end{aligned}$$

$$
\begin{aligned}
\langle res(U_H), w\rangle \;\leq\; & \sum_{T\in\mathcal{T}_H} c_1 h_T \|f+\nabla(D\nabla U_H)\|_{0,T}\|\nabla w\|_{0,\Omega_T} \\
& + \sum_{S'\in\mathcal{S'}_H} c_2 h_{S'}^{1/2}\|P_H - D\nabla U_H.n_{S'}\|_{0,S'}\|\nabla w\|_{0,\Omega_{S'}} \\
& + \sum_{S\in\mathcal{S}_H} c_2 h_S^{1/2}\|[\![D\nabla U_H.n_S]\!]_s\|_{0,S}\|\nabla w\|_{0,\Omega_S} \\
\leq\; & c\Big\{ \sum_{T\in\mathcal{T}_H} h_T^2\|f+\nabla(D\nabla U_H)\|_{0,T}^2 \\
& + \sum_{S'\in\mathcal{S'}_H} h_{S'}\|P_H - D\nabla U_H.n_{S'}\|_{0,S'}^2 \\
& + \sum_{S\in\mathcal{S}_H} h_S\|[\![D\nabla U_H.n_S]\!]_s\|_{0,S}^2\Big\}^{1/2} \\
& \cdot\Big\{ \sum_{T\in\mathcal{T}_H}\|\nabla w\|_{0,\Omega_T}^2 + \sum_{S\in\mathcal{S}_H\cup\mathcal{S'}_H}\|\nabla w\|_{0,\Omega_S}^2\Big\}^{1/2} \\
\leq\; & c_T.c|w|_{1,\Omega}\Big\{ \sum_{T\in\mathcal{T}_H} h_T^2\|f+\nabla(D\nabla U_H)\|_{0,T}^2 \\
& + \sum_{S'\in\mathcal{S'}_H} h_{S'}\|P_H - D\nabla U_H.n_{S'}\|_{0,S'}^2 \\
& + \sum_{S\in\mathcal{S}_H} h_S\|[\![D\nabla U_H.n_S]\!]_s\|_{0,S}^2\Big\}^{1/2}
\end{aligned}
\tag{2.46}
$$

donde c_T depende únicamente del mínimo ángulo de la triangulación.

Siguiendo a [54] obtenemos

$$
\begin{aligned}
|u-U_H| \leq & C\Big\{ \sum_{T\in\mathcal{T}_H} h_T^2\|f+\nabla(D\nabla U_H)\|_{0,T}^2 + \sum_{S'\in\mathcal{S'}_H} h_{S'}\|P_H - D\nabla U_H.n_{S'}\|_{0,S'}^2 \\
& + \sum_{S\in\mathcal{S}_H} h_S\|[\![D\nabla U_H.n_S]\!]_s\|_{0,S}^2\Big\}^{1/2}
\end{aligned}
\tag{2.47}
$$

Denotaremos por η_T al indicador local del error asociado a cada elemento $T \in \mathcal{T}_H$,

$$\eta_T^2 := h_T^2 \|f + \nabla(D\nabla U_H)\|_{0,T}^2 + \sum_{S' \in \mathcal{S'}_H} h_{S'} \|P_H - D\nabla U_H . n_{S'}\|_{0,S'}^2 + \frac{1}{2} \sum_{S \in \mathcal{S}_H} h_S \|[\![D\nabla U_H . n_S]\!]_s\|_{0,S}^2 \tag{2.48}$$

donde h_T representa el diámetro de cada elemento $T \in \mathcal{T}_H$ y h_s la longitud de cada arista. El estimador global del error η_H se define,

$$\eta_H^2 := \sum_{T \in \mathcal{T}_H} \eta_T^2 \tag{2.49}$$

Lema 2.1. ***(Estimación Superior del error)*** *Existe una constante C_1, que sólo depende del ángulo mínimo de $\mathcal{T}_H$ tal que,*

$$\sum_{i=1}^{nd} |u^i - U_H^i|_{1,\Omega_i}^2 \leq C_1 \eta_H^2 = C_1 (\sum_{T \in \mathcal{T}_H} \eta_T^2)$$

2.7. Convergencia del AMUADD

En este apartado demostraremos que el AMUADD, propuesto en la sección §2.4 es convergente, es decir, genera una sucesión de soluciones discretas $\{(U_j, P_j)\}$ para $j \geq 0$ tales que,

$$\|u - U_j\|_{\mathbb{V}} + \|p - P_j\|_{\mathbb{M}} \leq C\delta^j \tag{2.50}$$

para una constante $C > 0$ independiente de $j, f, P_{j-1}, \mathbb{V}_j$ y $0 < \delta < 1$. Para ello, hemos de suponer que nuestro algoritmo, con la estrategia de refinamiento que consideramos, satisface la siguiente condición

$$\|u_j - U_j\|_{\mathbb{V}} \leq C\varepsilon_j \tag{2.51}$$

que expresa la convergencia del procedimiento que hemos utilizado en el cálculo de las aproximaciones U_j.

Observación.

- El resultado anterior se consigue gracias a la aplicación de un MEFA basado en la estimación a-posteriori de tipo residual que asegura la convergencia de la secuencia $\{U_j\}_{j\geq 0}$ a la solución u_j de (2.28), (ver [42, 43]).

La convergencia del AMUADD está dada en forma de teorema que presentamos y demostramos a continuación:

Teorema 2.2. *Sea $\rho > 0$ verificando (2.25), $0 < \theta < 1$, y el procedimiento definido por nuestro algoritmo en el cálculo de $U_j \in \mathbb{V}_j$. Entonces existen constantes positivas, C y $\delta < 1$, tales que las soluciones (U_j, P_j) generadas por* AMUADD *verifican,*

$$\|u - U_j\|_{\mathbb{V}} + \|p - P_j\|_{\mathbb{M}} \leq C\delta^j$$

donde $(u, p) \in \mathbb{V}_J \times \mathbb{M}_J$ son solución del Problema 2.2, donde se han substituido $\mathbb{V}$ y $\mathbb{M}$ por $\mathbb{V}_J$ y $\mathbb{M}_J$.

Demostración. En cada etapa $j \geq 1$ se resuelve,

$$U_j \in \mathbb{V}_j : \qquad a(U_j, V) = \langle f, V\rangle - (P_{j-1}, BV), \quad \forall V \in \mathbb{V}_j \tag{2.52}$$

que aproxima el problema,

$$u_j \in \mathbb{V} : \qquad a(u_j, v) = \langle f, v\rangle - \langle B^* P_{j-1}, v\rangle \qquad \forall v \in \mathbb{V} \tag{2.53}$$

es decir

$$u_j \in \mathbb{V} : \qquad Au_j = f - B^* P_{j-1}, \quad \text{en } \mathbb{V}^* \tag{2.54}$$

Eliminando u_j en (2.54) y considerando la definición del operador S, en D.2.3, se tiene,

$$u_j \in \mathbb{V}: \qquad Bu_j = BA^{-1}f - SP_{j-1}, \quad \in \mathbb{M} \tag{2.55}$$

y recordemos que p es solución de la ecuación

$$p = (I - \rho S)p + \rho BA^{-1}f \tag{2.56}$$

Sean U_j y P_j definidos en (2.30) y (2.35). Sumando y restando ρBu_j en (2.35) tendremos que,

$$P_j = P_{j-1} + \rho B(U_j - u_j) + \rho Bu_j$$

Como se verifica la igualdad (2.55), substituyendo ρBu_j en la expresión anterior tenemos

$$P_j = (I - \rho S)P_{j-1} + \rho B(U_j - u_j) + \rho BA^{-1}f \tag{2.57}$$

Restando (2.56) y (2.57) obtenemos,

$$p - P_j = (I - \rho S)(p - P_{j-1}) - \rho B(U_j - u_j) \tag{2.58}$$

aplicando la desigualdad triangular y poniendo $\sigma := \|I - \rho S\|_{\mathcal{L}(M,M)}$ se tiene,

$$\|p - P_j\|_{\mathbb{M}} \leq \sigma\|p - P_{j-1}\|_{\mathbb{M}} + \rho\|B(U_j - u_j)\|_{\mathbb{M}} \tag{2.59}$$

Como B es un operador lineal y continuo, existe una $C_1 > 0$ tal que

$$\|Bv\|_{\mathbb{M}} \leq C_1\|v\|_{\mathbb{V}} \qquad \forall v \in \mathbb{V} \tag{2.60}$$

donde

$$\|p - P_j\|_{\mathbb{M}} \leq \sigma\|p - P_{j-1}\|_{\mathbb{M}} + \rho C_1\|U_j - u_j\|_{\mathbb{V}} \tag{2.61}$$

Utilizando la condición (2.51),

$$\begin{aligned}\|p - P_j\|_{\mathbb{M}} &\leq \sigma\|p - P_{j-1}\|_{\mathbb{M}} + C\varepsilon_j \leq \\ &\leq \sigma\|p - P_{j-1}\|_{\mathbb{M}} + C\varepsilon_0\gamma^j\end{aligned}$$

Por inducción se obtiene,

$$\|p - P_j\|_{\mathbb{M}} \leq \sigma^j\|p - P_0\|_{\mathbb{M}} + C\varepsilon_0\sum_{l=0}^{j-1}\sigma^l\gamma^{j-l}$$

y tomando $\vartheta := \text{máx}\{\sigma, \gamma\}$, se sigue que,

$$\|p - P_j\|_{\mathbb{M}} \leq \vartheta^j\|p - P_0\|_{\mathbb{M}} + Cj\vartheta^j \tag{2.62}$$

Si $\delta > 0$ es tal que $\vartheta < \delta < 1$ entonces $j\vartheta^j < \delta^j$ para j suficientemente grande, ($xa^x \geq 0$, $\lim_{x\to\infty} x \cdot a^x = 0$, para $0 < a < 1$) y se tiene,

$$\|p - P_j\|_{\mathbb{M}} \leq C\delta^j \tag{2.63}$$

para ciertas constantes positivas C y $\vartheta < \delta < 1$. Para obtener una cota similar para $\|u - U_j\|_{\mathbb{V}}$, observar que,

$$\|u - u_j\|_{\mathbb{V}} \leq C\|p - P_{j-1}\|_{\mathbb{M}}$$

entonces,

$$\|u - U_j\|_{\mathbb{V}} \leq \|u - u_j\|_{\mathbb{V}} + \|U_j - u_j\|_{\mathbb{V}} \leq C\|p - P_{j-1}\|_{\mathbb{M}} + C\varepsilon_j$$

y se concluye con (2.63). □

Capítulo 3

Ejemplos Numéricos

El objetivo de este capítulo es mostrar los resultados obtenidos por aplicación de nuestro algoritmo adaptativo a un problema con una singularidad en una región bien definida en el dominio original. Vamos a considerar distintas descomposiciones de dominios, concretamente descomposiciones en dos, tres y cuatro subdominios y confrontaremos los resultados con los obtenidos en un solo dominio, en el dominio original, sin adaptación de mallado.

Para la resolución de los experimentos numéricos hemos desarrollado un programa de elementos finitos a partir de la biblioteca básica neptuno++ de L. Ferragut, [21] implementada en lenguaje *C++* y que llamamos neptunoDD++.

Para la representación gráfica de los resultados se ha usado el software de visualización MEDIT (ver [22]).

Para comparar el comportamiento de los errores y estimadores en los distintos ejemplos y puesto que utilizamos mallas adaptadas que pueden llegar a ser muy diferentes en magnitud en cada subdominio o en distintas zonas de un mismo subdominio, relacionamos el error con el número de grados de libertad, DOF.

Se espera que el comportamiento para el error,

$$\mathbf{e}_j := |u - U_j| \approx C\ \mathtt{DOF}_j^{-\frac{1}{2}}.$$

El indicador numérico para este resultado es el denominado Orden de Convergencia Experimental en la iteración j-ésima $\mathtt{EOC}_j$,

$$\mathtt{EOC}_j = -2\frac{\log(\mathbf{e}_j/\mathbf{e}_{j-1})}{\log(\mathtt{DOF}_j/\mathtt{DOF}_{j-1})}.$$

que debería tender asintóticamente hacia 1.

En cada ejemplo mostraremos los mallados iniciales y finales obtenidos en cada uno de los subdominios de la descomposición considerada, y también los gráficos para la solución aproximada en cada uno de ellos y de los multiplicadores, así como tablas donde aparecerán reflejados datos de interés del AMUADD. Completaremos el estudio con curvas de error (en la norma $H^1_{0,\Gamma_l}(\Omega_l)$) frente al número de grados de libertad.

La notación empleada en las tablas de resultados es la siguiente: la columna `iter` denota el número de iteración (exterior) del AMUADD, la columna $\mathtt{R.A.}\Omega_i$ se refiere al número de refinamientos acumulados en cada iteración de Uzawa para cada subdominio, necesarios para obtener la convergencia. También se muestran los errores de Uzawa, es decir el error en la norma $\mathbb{M}$ del multiplicador. El hecho que en las tablas aparezcan los subíndices i, j asociados a esta norma es para identificar de forma sencilla el multiplicador ya que podemos llegar a tener cuatro multiplicadores diferentes, dependiendo de la descomposición considerada. Es decir, $\mathbb{M}_{ij}$ representa el espacio donde está el multiplicador definido en frontera interna entre los subdominios Ω_i y Ω_j. Para el cálculo de las normas de los errores usaremos cuadraturas de Gauss con 2 puntos (exactas para polinomios de grado 3) ya que esta nos permiten obtener una precisión aceptable.

Para ilustrar la aplicación del Algoritmo de Uzawa Modificado con Descomposición de Dominios (AMUADD) se considera un problema estacionario con una capa límite interior que se resuelve considerando diferentes descomposiciones del dominio inicial Ω. Notar que en los ejemplos que presentamos a continuación no se refinan adaptativamente las mallas de las fronteras interiores Γ_{lk}. Pero ya se han programado las funciones necesarias para que en el futuro se pueda incorporar al (AMUADD) la adaptación de mallados en las fronteras Γ_{lk}, que resultan de la descomposición del dominio considerada.

El siguiente problema se ha elegido porque presenta una capa límite interna de la que a-priori no sabemos exactamente donde está localizada sin conocer previamente la solución.

El problema viene dado por,

Problema 3.1.

$$-\nabla(a(x,y)\nabla u) = f(x,y) \text{ en } \Omega = [0,1]^2$$
$$u(x,0) = u(0,y) = u(x,1) = u(1,y) = 0$$

donde

$$a(x,y) = \frac{1}{\alpha} + \alpha(r-\beta)^2$$
$$f(x,y) = 2[1 + \alpha(r-\beta)\arg\tan(\alpha(r-\beta) + \arg\tan\alpha r)]$$

con $\quad r = \sqrt{x^2+y^2}, \quad \alpha = 100 \quad$ y $\quad \beta = 0.36388.$

En los ejemplos que se presentan a continuación se consideran los siguientes valores para los párametros:

$$\rho = 25, \quad \varepsilon_0 = 1.0, \quad \gamma = 0.6$$

3.1. Ejemplo 1

En este ejemplo se considera una descomposición del dominio inicial Ω en dos subdominios $\Omega_1 = [0, 0.5] \times [0, 1]$ y $\Omega_2 = [0.5, 1] \times [0, 1]$.

Al considerar esta descomposición obtenemos dos subdominios con la misma área pero con la capa límite interior situada casi completamente en el interior del subdominio Ω_1. Se considera una malla inicial (ver Figura 3.1), $\mathcal{T}_0$, con 66 grados de libertad (DOF), en cada uno de los subdominios. Para resolver el problema definido en la frontera interna Γ_{12} se considera un mallado con 1001 grados de libertad (DOF) y la longitud de cada arista $\mathcal{S}'$ es $h_{\mathcal{S}'} = 0.001$.

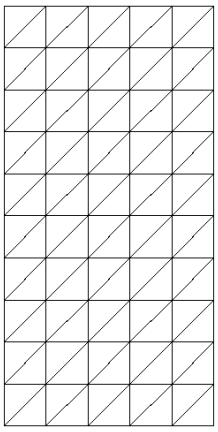


Figura 3.1: Malla inicial de Ω_1 y Ω_2

En la Figura 3.2 presentamos las soluciones finales aproximadas U_1 y U_2

obtenidas tras 25 iteraciones en Ω_1 y Ω_2, así como los correspondientes mallados donde han resultado dichas aproximaciones. El mallado final $\mathcal{T}_n$ en Ω_1 tiene 23012 grados de libertad (DOF) mientras el de Ω_2 tiene 38205 grados de libertad (DOF).

El hecho de que se haya refinado más la malla de Ω_2 se explica por el error asociado a cada elemento de la triangulación es del mismo orden de magnitud, por lo que se refinan muchos elementos del mallado en cada iteración. En cambio en el subdominio Ω_1 la malla se refina más cerca de la capa límite que es donde están los elementos que presentan mayor error en la solución. Además se observa la continuidad de la solución entre los dos subdominios.

En la Figura 3.3 se representan los indicadores de error η_{U_i} en cada solución U_i, $i = 1, 2$ y $\eta_U := \eta_{U_1} + \eta_{U_2}$ define el indicador de error en la solución global $U = U_1 + U_2$. Comparando las pendientes de las rectas que representan los indicadores de error con las rectas de convergencia óptima, se observa un comportamiento quasi-óptimo en cada uno de los casos.

Se ha resuelto el mismo problema sin adaptación de mallado, considerando diferentes grados de libertad, calculamos los indicadores de error $\eta_U := \eta_{U_1} + \eta_{U_2}$ y para esos mismos valores calculamos la correspondiente recta de convergencia óptima (ver Figura 3.4). Comparando las pendientes de las rectas que representan los indicadores de error con y sin adaptación de mallado se constata que la pendiente de la Figura 3.4, que corresponde a no hacer adaptación de mallado, es más horizontal que la pendiente de la gráfica de la Figura 3.3 que ha resultado de la aplicación del AMUADD. Por lo que se puede concluir que la convergencia con adaptación de mallado es más rápida y además la precisión en los resultados obtenidos es mayor cuando se resuelve el problema con adaptación de mallado.

Respecto al tiempo de cálculo, se verifica que el algoritmo sin adaptación de

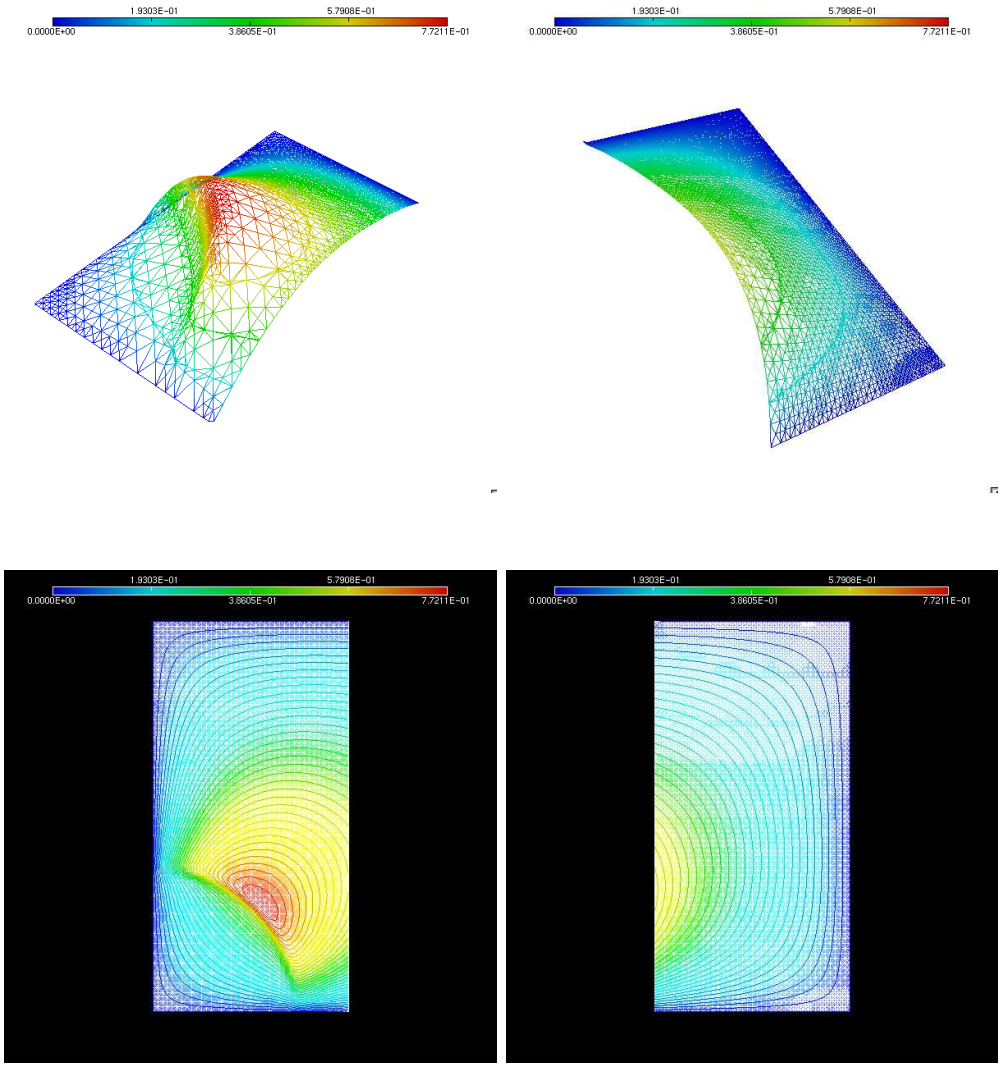


Figura 3.2: Solución en Ω_1(izquierda) y en Ω_2(derecha)

mallado, con un total de 59858 grados de libertad, tarda 2310.57 segundos, en cambio, en la resolución de problema con el AMUADD se tarda 228.66 segundos.

El comportamiento del error de Uzawa (ver Figura 3.5) nos permite concluir

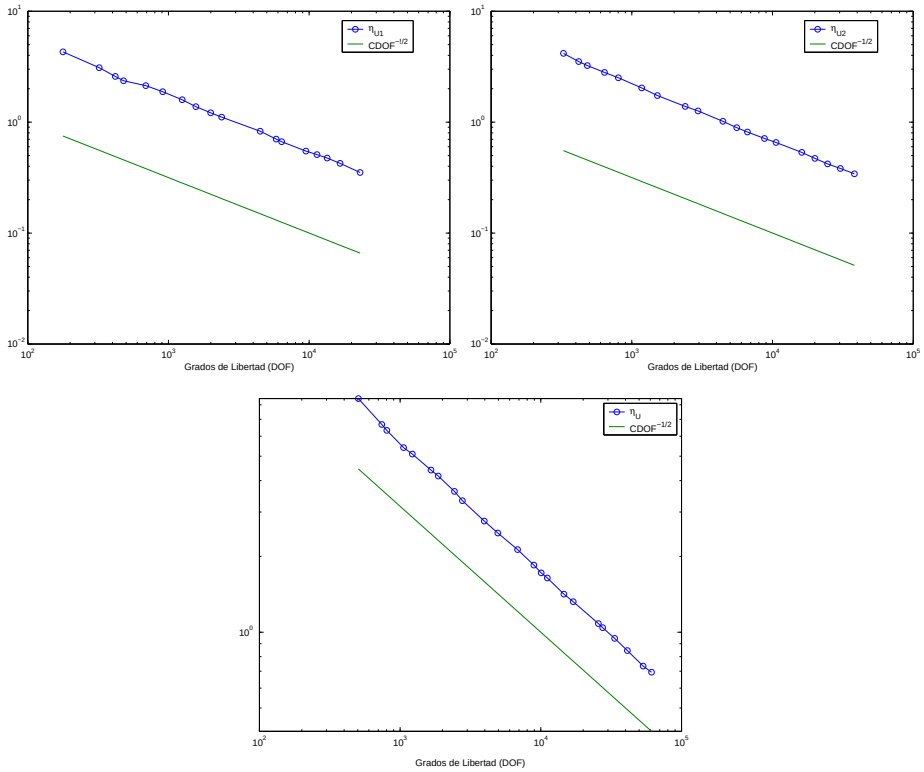


Figura 3.3: Error en Ω_1 (izquierda), en Ω_2(derecha) y Error global (abajo)

que se tiene la convergencia en el multiplicador y además, si comparamos los errores obtenidos por aplicación de nuestro algoritmo con y sin adaptación de mallado (para DOF=10201), constatamos que claramente son mejores los resultados con adaptación de mallado.

En la Tabla 3.1 se presentan los resultados obtenidos para $n = 1, ..., 25$. Observar cómo el número de refinamientos en cada iteración de Uzawa es a lo sumo 3 en Ω_1, 1 en Ω_2 y que se tiene la convergencia del multiplicador.

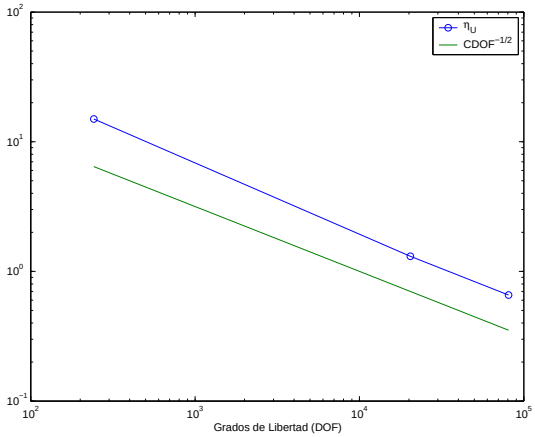


Figura 3.4: Error global sin adaptación de mallado

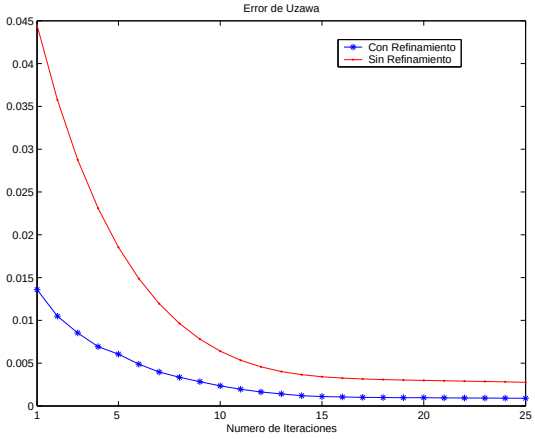


Figura 3.5: Error de Uzawa con y sin adaptación de mallado

Iter	R.A. Ω_1	R.A. Ω_2	DOF1	DOF2	$\|P_j - P_{j-1}\|_{\mathbb{M}}$
1	3	6	178	327	1.3159e-02
2	4	7	322	419	1.0496e-02
3	4	7	322	419	8.5376e-03
4	4	8	322	483	6.9296e-03
5	5	9	419	640	6.0538e-03
6	5	10	419	802	4.8826e-03
7	6	11	480	1177	3.9630e-03
8	7	11	690	1177	3.3479e-03
9	9	12	911	1521	2.8342e-03
10	11	12	1251	1521	2.3431e-03
11	12	14	1566	2396	1.9495e-03
12	12	14	1566	2396	1.6333e-03
13	15	15	1996	2954	1.4073e-03
14	16	16	2390	4443	1.1953e-03
15	19	16	4488	4443	1.1053e-03
16	19	17	4448	5574	1.0447e-03
17	19	18	4448	6636	1.0067e-03
18	20	19	5830	8772	9.8448e-04
19	21	20	6377	10600	9.6679e-04
20	23	21	9479	16144	9.6367e-04
21	24	21	11377	16144	9.4727e-04
22	27	22	13432	20032	9.3491e-04
23	28	23	16598	24622	9.2424e-04
24	30	24	23012	30351	9.0981e-04
25	30	25	23012	38205	8.9458e-04

Tabla 3.1: Tabla de resultados para la descomposición en dos subdominios simétricos. Notación: `iter`, iteración del AMUADD; R.A.Ω_i, número de refinamientos acumulados en Ω_i; `DOFi`, número de grados de libertad en Ω_i; $\|P_j - P_{j-1}\|_{\mathbb{M}}$, error de Uzawa.

3.2. Ejemplo 2

A continuación presentamos los resultados obtenidos del problema considerando una descomposición en dos subdominios no simétricos, es decir, $\Omega_1 = [0, 0.3] \times [0, 1]$ y $\Omega_2 = [0.3, 1] \times [0, 1]$.

Con esta descomposición el subdominio Ω_1 tiene menor área que el subdominio Ω_2, y la capa límite ahora queda repartida entre los dos sudominios. Al principio los mallados iniciales $\mathcal{T}_0$ considerados en Ω_1 tienen 66 grados de libertad (DOF) y en Ω_2 88 grados de libertad (DOF)(ver Figura 3.6). Para resolver el problema definido en la frontera interna Γ_{12} se considera un mallado con 1001 grados de libertad (DOF) y la longitud de cada arista $\mathcal{S}'$ es $h_{\mathcal{S}'} = 0.001$.

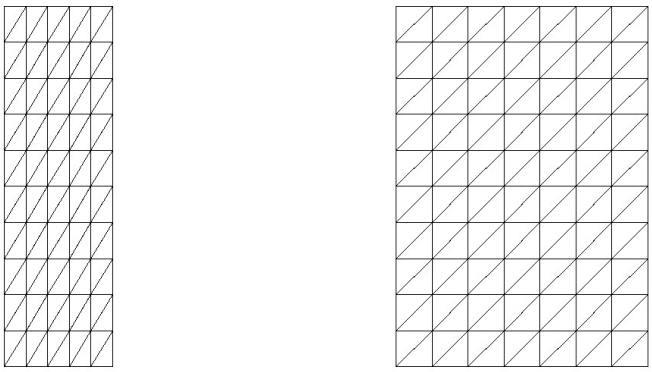


Figura 3.6: Mallas iniciales en Ω_1(izquierda) y en Ω_2(derecha)

En la Figura 3.7 presentamos, arriba las soluciones y mallados finales obtenidos en cada subdominio, y abajo otra representación de las soluciones. Se observa como se consigue la continuidad de las soluciones obtenidas en los dos subdo-

minios. El mallado $\mathcal{T}_n$ en Ω_1 tiene 26007 grados de libertad (DOF) mientras que el de Ω_2 tiene 30086 grados de libertad (DOF). El mallado en Ω_2 sigue siendo más fino que en Ω_1 y los elementos más pequeños de la triangulación se concentran alrededor de la capa límite. Se sigue refinado más la malla de Ω_2, pero si comparamos con el mallado final obtenido en el la descomposición del ejemplo anterior, el número de grados de libertad aquí es menor porque Ω_2 tiene una parte significativa de la capa límite.

En la Figura 3.8 se representan los indicadores de error η_{U_i} en cada solución U_i, $i = 1, 2$ y $\eta_U := \eta_{U_1} + \eta_{U_2}$ define el indicador de error en la solución global $U = U_1 + U_2$. Comparado con la recta de convergencia óptima se observa un comportamiento quasi-óptimo en cada uno de los casos.

Se presentan a continuación los resultados obtenidos con la resolución del problema usando la misma descomposición pero sin adaptación de mallado. En ningúno de los casos, con 20402, 56026 y 83642 grados de libertad (DOF), se consigue en Ω_2 convergencia a la solución y los tiempos de cálculo en las 25 iteraciones son respectivamente 137.14, 535.8 y 999.33 segundos. En cambio, con adaptación de mallado, en la última iteración se tiene un total de 61217 grados de libertad y el algoritmo converge para la solución en cada subdominio al final de 228.66 segundos. Es decir, con la descomposición considerada el AMUADD consigue solucionar el problema, mientras que sin adaptación nos es posible solucionarlo, aunque se considere un mallado muy fino.

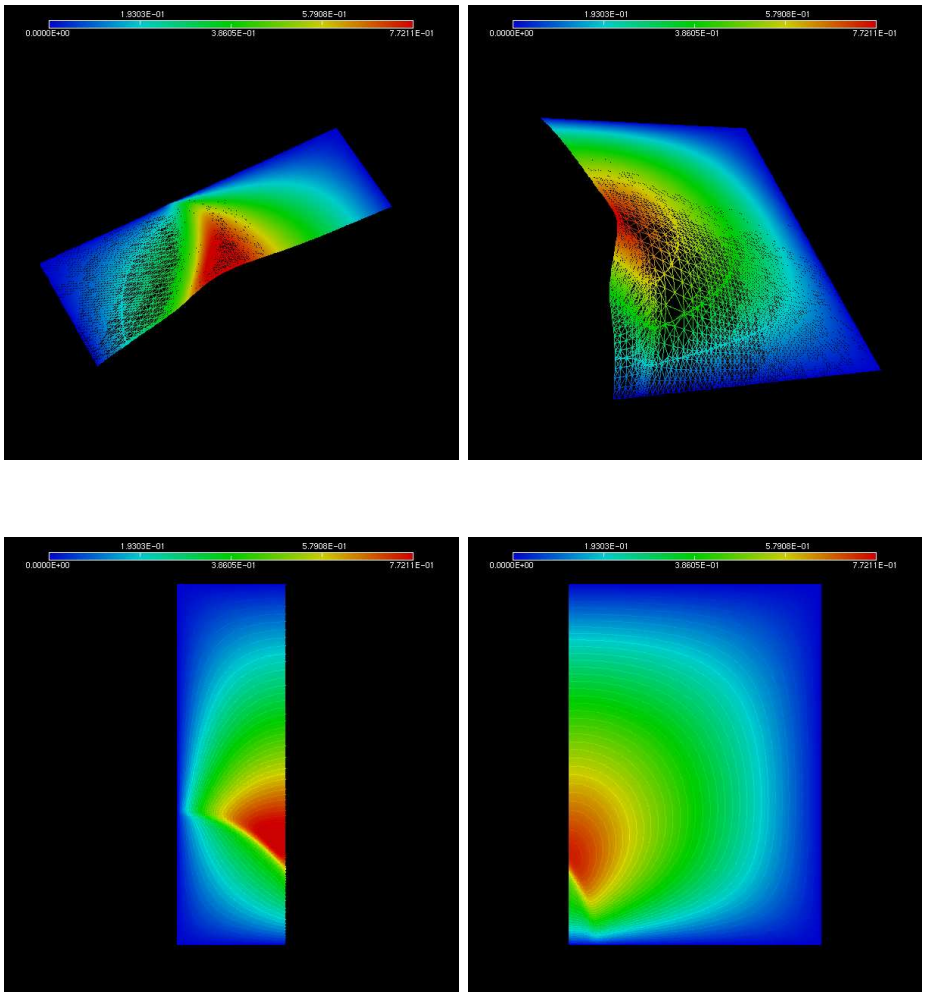


Figura 3.7: Solución en Ω_1(izquierda) y en Ω_2(derecha)

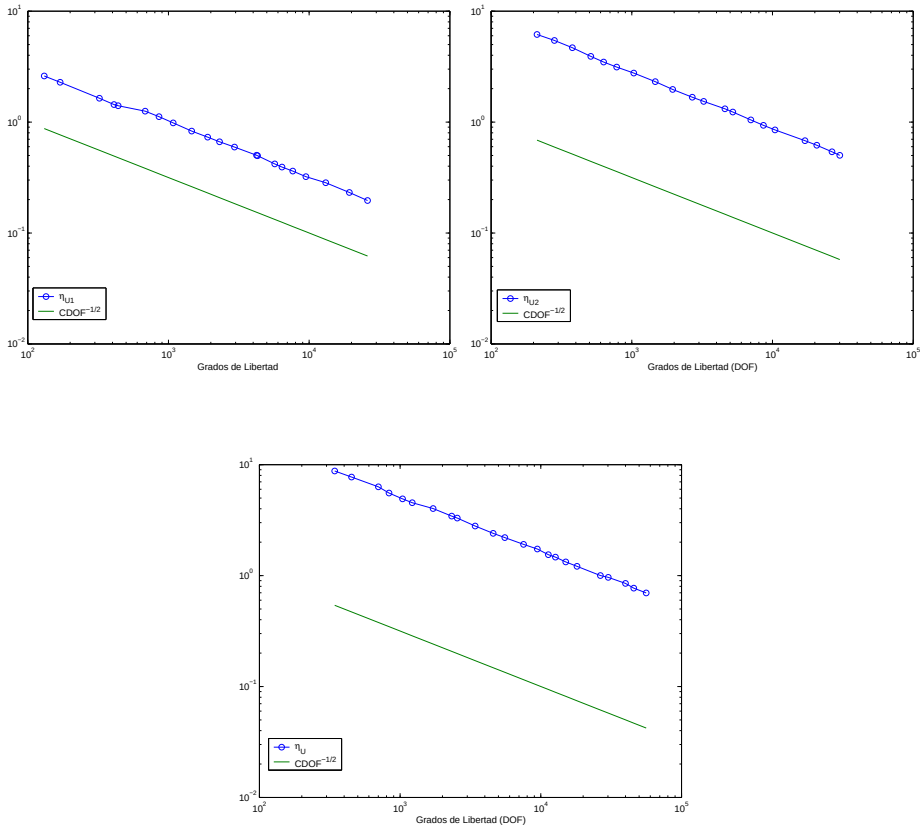


Figura 3.8: Error en Ω_1 (izquierda), en Ω_2(derecha) y Error global(abajo)

En la Figura 3.9 se presenta la gráfica del comportamiento del error de Uzawa con adaptación de mallado. Se observa como el error disminuye con el número de iteraciones, es decir, se tiene la convergencia en el multiplicador.

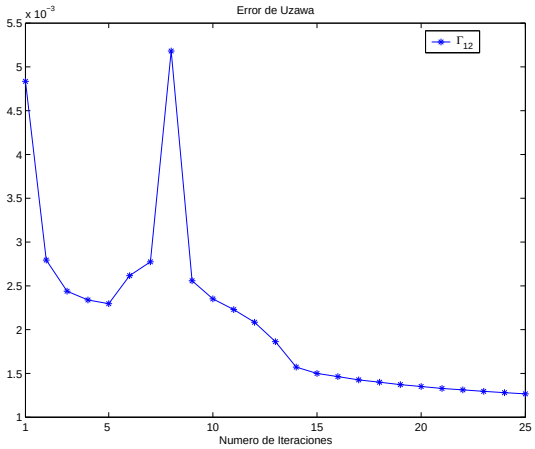


Figura 3.9: Error de Uzawa

En la Tabla 3.2 se presentan los resultados obtenidos para n=1,...,25. Observar cómo el número de refinamientos en cada iteración de Uzawa es a lo sumo 5 en Ω_1, 3 en Ω_2 y que se tiene la convergencia del multiplicador.

Iter	R.A. Ω_1	R.A. Ω_2	DOF1	DOF2	$\|P_j - P_{j-1}\|_{\mathbb{M}}$
1	2	4	131	212	4.8347e-03
2	4	5	170	282	2.79416e-03
3	5	6	323	378	2.4391e-03
4	5	7	323	511	2.3386e-03
5	5	7	323	511	2.2956e-03
6	6	8	410	631	2.6167e-03
7	7	9	439	781	2.7741e-03
8	10	10	683	1033	5.18121e-03
9	11	11	857	1469	2.5580e-03
10	12	11	1080	1469	2.3511e-03
11	13	13	1463	1952	2.2288e-03
12	13	13	1463	1952	2.0844e-03
13	17	15	1899	2689	1.8628e-03
14	19	16	2308	3243	1.5719e-03
15	20	18	2947	4591	1.5001e-03
16	22	19	4226	5226	1.4640e-03
17	23	20	4301	7016	1.4259e-03
18	24	20	5696	7016	1.3995e-03
19	26	21	6411	8621	1.3720e-03
20	28	22	7652	10420	1.3508e-03
21	29	24	9481	17029	1.3282e-03
22	34	24	13122	17029	1.3120e-03
23	35	25	19316	20658	1.2954e-03
24	35	28	19316	26462	1.2808e-03
25	39	30	26007	30086	1.2661e-03

Tabla 3.2: Tabla de resultados para la descomposición en dos subdominios no simétricos. Notación: `iter`, iteración del AMUADD; `R.A.`Ω_i, número de refinamientos acumulados en Ω_i; `DOFi`, número de grados de libertad en Ω_i; $\|P_j - P_{j-1}\|_{\mathbb{M}}$, error de Uzawa.

3.3. Ejemplo 3

A continuación presentamos los resultados obtenidos de la resolución del problema para la descomposición del dominio Ω en tres subdominios, $\Omega_1 = [0, 0.3] \times [0, 1]$, $\Omega_2 = [0.3, 0.6] \times [0, 1]$ y $\Omega_3 = [0.6, 1] \times [0, 1]$. Al igual que en el Ejemplo 3.2 la capa límite está repartida entre los subdominios Ω_1 y Ω_2. Con esta descomposición obtenemos, Ω_1 y Ω_2 con las mismas dimensiones y el subdominio Ω_3 un poco más grande que los otros dos y se consideran dos multiplicadores de Lagrange porque se tienen dos fronteras internas, Γ_{12} entre Ω_1 y Ω_2 y Γ_{23} entre Ω_2 y Ω_3. En Ω_i, $i = 1, 2$, se consideran mallados iniciales iguales, a saber $\mathcal{T}_0$ con 44 grados de libertad (DOF) y $\mathcal{T}_0$ en Ω_3 con 55 grados de libertad (DOF)(ver Figura 3.10). Para resolver los problemas definidos en las fronteras internas Γ_{12} y Γ_{23} se consideran en ambos casos un mallado con 1001 grados de libertad (DOF) y la longitud de cada arista $\mathcal{S}'$ es $h_{\mathcal{S}'} = 0.001$.

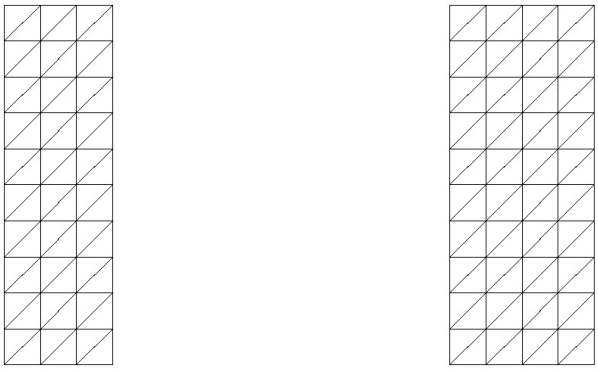


Figura 3.10: Mallas iniciales en Ω_1 y Ω_2 (izquierda) y en Ω_3(derecha)

En la Figura 3.11 presentamos las soluciones y mallas finales obtenidas en cada subdominio. El mallado $\mathcal{T}_n$ tiene 22648, 27811 y 40680 grados de libertad (DOF) en Ω_1, Ω_2 y Ω_3 respectivamente. El número de refinamientos efectuados en Ω_1 y Ω_2 es prácticamente el mismo, siendo en Ω_3 donde el número de adaptaciones del mallado es menor, sin embargo, como los errores de los elementos son del mismo orden de magnitud, la malla obtenida es más fina que en los otros dos subdominios.

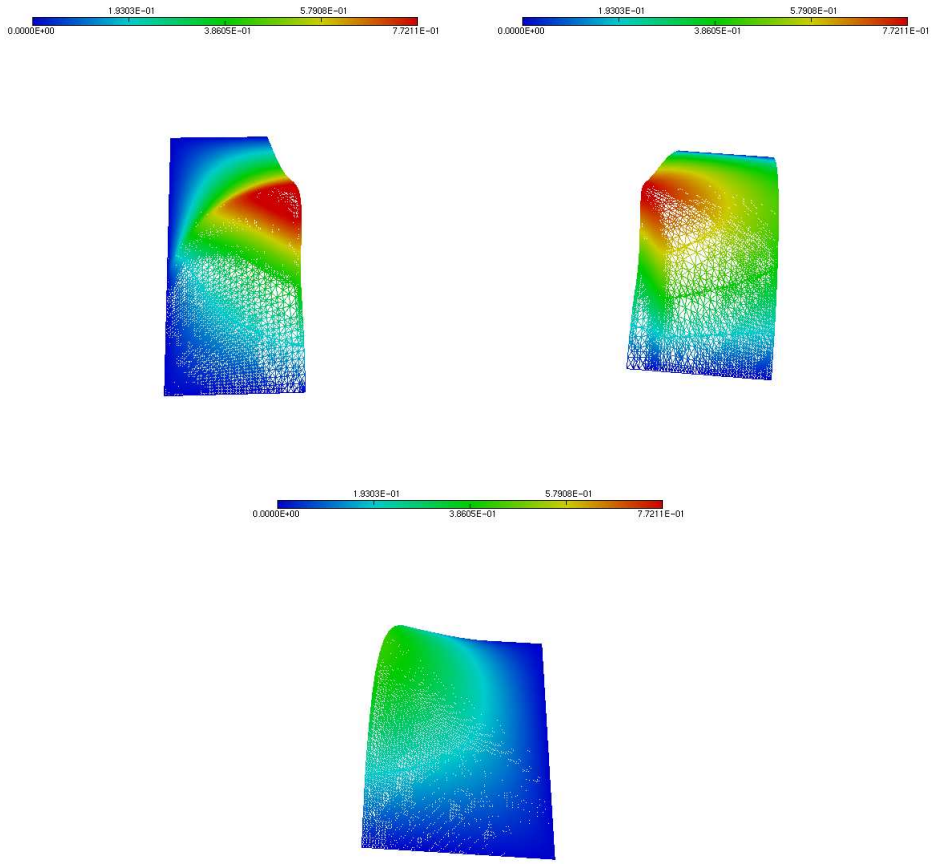


Figura 3.11: Solución en Ω_1(izquierda), Ω_2(derecha) y Ω_3(abajo)

En la Figura 3.12 presentamos la solución de otra forma distinta, aquí se puede ver las isolíneas que definen las temperaturas de la solución y cómo se consigue la continuidad de la solución entre los 3 subdominios.

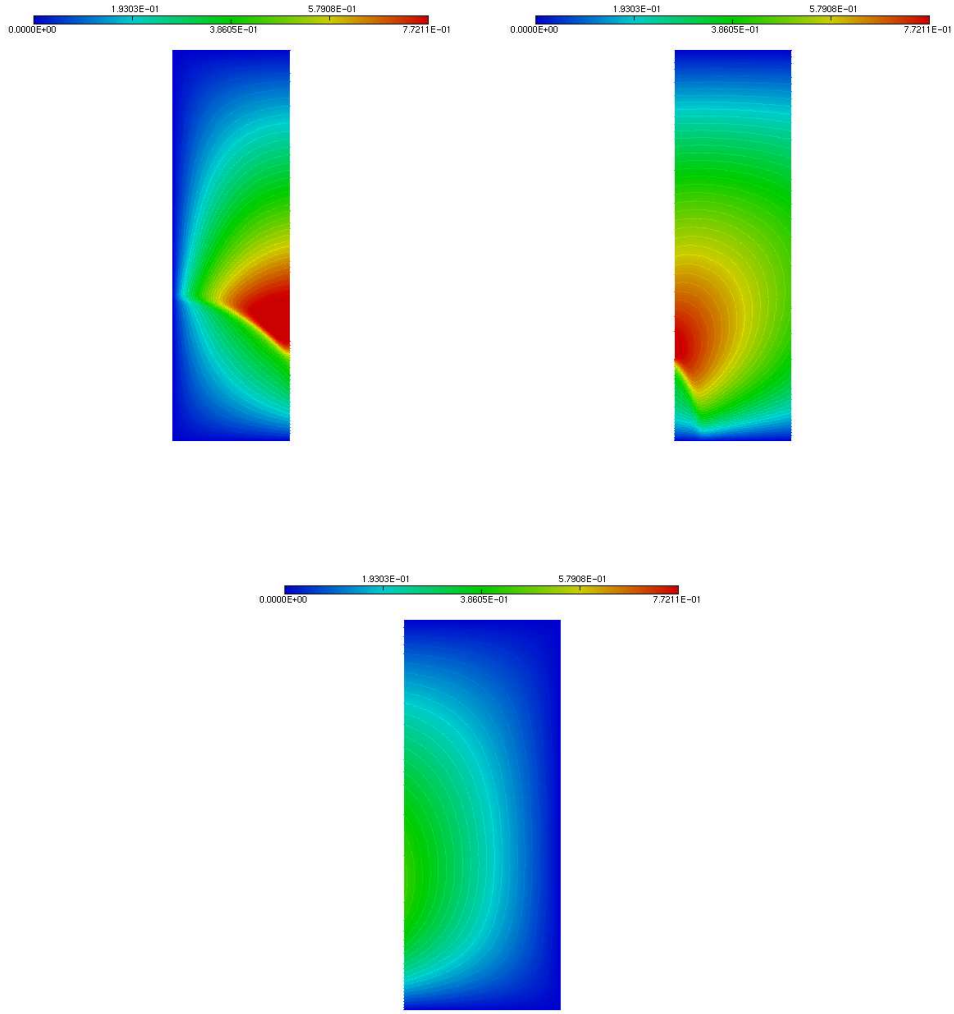


Figura 3.12: Solución en Ω_1(izquierda), Ω_2(derecha) y Ω_3(abajo)

En la Figura 3.13 se puede ver el comportamiento del error a-posteriori en cada uno de los subdominios $\eta_{U_i}, i = 1, 2, 3$ y en la Figura 3.14 en la gráfica de la izquierda el comportamiento del estimador del error global a-posteriori $\eta_U := \eta_{U_1} + \eta_{U_2} + \eta_{U_3}$. Comparando el estimador del error obtenido con la recta de convergencia óptima, se observa en cada uno de los casos un comportamiento quasi-óptimo. El subdominio Ω_1 sigue siendo aquel donde la convergencia a la solución se hace más rápidamente y donde se tiene mayor precisión en la solución. En la Figura 3.14 (gráfica derecha), presentamos el comportamiento del error de Uzawa de ambos multiplicadores. En cada caso el valor del error decrece con el número de iteraciones, pero el error del algoritmo de Uzawa en el multiplicador definido en Γ_{23} es menor y la curva decrece de una forma más regular.

En la Tabla 3.3 se presentan los resultados obtenidos en las 25 iteraciones realizadas. Obsérvese cómo el número de refinamientos en cada iteración es a lo sumo 5 en Ω_1, 7 en Ω_2 y 2 en Ω_3. En las dos últimas columnas de la Tabla 3.3 se puede ver como se tiene la convergencia de los multiplicadores.

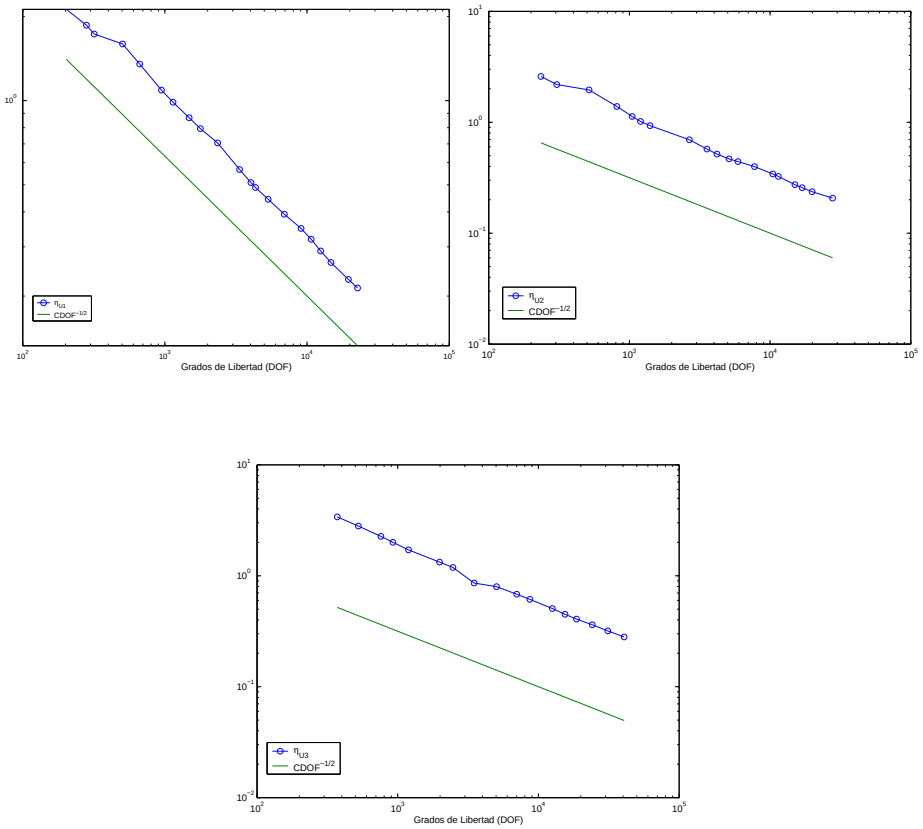


Figura 3.13: Indicadores del error en Ω_1 (izquierda), en Ω_2 (Derecha) y en Ω_3 (abajo)

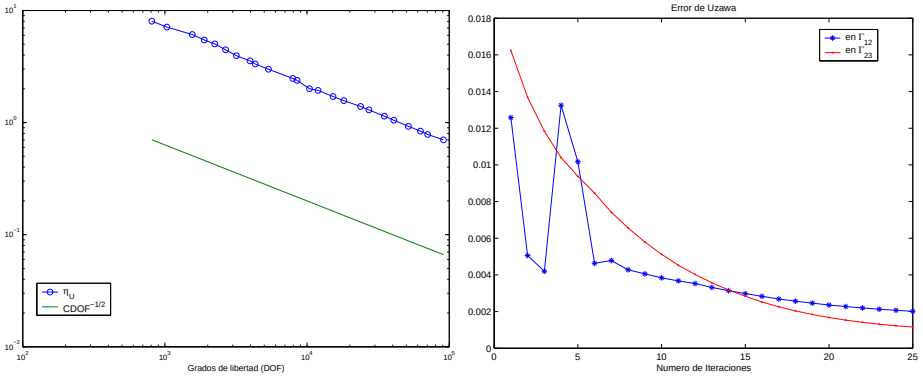


Figura 3.14: Error Total y Error de Uzawa

Iter	R.A.Ω_1	R.A.Ω_2	R.A.Ω_3	DOF1	DOF2	DOF3	$\|P_j - P_{j-1}\|_{\mathbb{M}_{12}}$	$\|P_j - P_{j-1}\|_{\mathbb{M}_{23}}$
1	5	5	7	203	235	372	1.2583e-02	1.62386e-02
2	5	6	8	203	305	525	5.0585e-03	1.36866e-02
3	5	6	8	203	305	525	4.19075e-03	1.18349e-02
4	7	13	9	281	510	759	1.01703e-02	1.03962e-02
5	11	14	9	446	802	759	1.01703e-02	9.38093e-03
6	12	14	10	503	802	925	4.63258e-03	8.46443e-03
7	13	14	11	858	802	1193	4.78578e-03	7.41312e-03
8	13	15	11	858	1193	1193	4.27838e-03	6.56251e-03
9	13	15	13	858	1193	1983	4.05468e-03	5.79183e-3
10	14	16	13	1023	1201	1983	3.84096e-03	5.12509e-3
11	18	17	14	1506	1399	2468	3.67438e-03	4.52338e-3
12	19	19	15	1712	2642	3491	3.52644e-03	4.02639e-3
13	20	19	15	2338	2642	3491	3.31418e-03	3.57214e-3
14	21	20	16	3291	3464	4345	3.14076e-03	3.17872e-3
15	21	20	17	3291	3464	5041	2.97971e-03	2.83334e-3
16	22	21	18	3942	4155	7018	2.83119e-03	2.52005e-3
17	23	23	19	4355	5075	8691	2.68613e-03	2.25795e-3
18	25	24	20	5867	5849	12566	2.56361e-03	2.03427e-3
19	26	26	20	6776	7655	12566	2.45918e-03	1.842070e-3
20	30	29	21	9867	10367	15450	2.35753e-03	1.67614e-3
21	32	30	22	13621	12189	18646	2.27001e-03	1.53186e-3
22	32	32	23	13621	15443	24128	2.1966e-03	1.41189e-3
23	33	33	24	14568	16808	31123	2.12677e-03	1.31145e-3
24	36	34	24	20879	19929	31123	2.0653e-03	1.22583e-3
25	38	39	25	22805	27370	40678	2.01374e-03	1.15519e-3

Tabla 3.3: Tabla de resultados para la descomposición en tres subdominios. Notación: `iter`, iteración del AMUADD; R.A.Ω_i, número de refinamientos acumulados en Ω_i; `DOFi`, número de grados de libertad en Ω_i; $\|P_j - P_{j-1}\|_{\mathbb{M}_{ij}}$, error de Uzawa en la frontera Γ_{ij}.

3.4. Ejemplo 4

3.4.1. Ejemplo 4 ($h_{\mathcal{S}'} = 0.001$)

A continuación presentamos los resultados obtenidos considerando ahora una descomposición del dominio inicial Ω en cuatro subdominios, $\Omega_1 = [0, 0.5] \times [0, 0.5]$, $\Omega_2 = [0.5, 1] \times [0, 0.5]$, $\Omega_3 = [0, 0.5] \times [0.5, 1]$ y $\Omega_4 = [0.5, 1] \times [0.5, 1]$. Con esta descomposición se definen 4 subdominios con la misma área y la capa límite está involucrada en el subdominio Ω_1. Para la resolución del problema en cada Ω_i, $i = 1, 2, 3, 4$ se considera un mallado inicial con 36 grados de libertad (DOF)(ver Figura 3.15) y cuatro multiplicadores de Lagrange definidos sobre las fronteras internas, Γ_{12}, entre Ω_1 y Ω_2, Γ_{13}, entre Ω_1 y Ω_3, Γ_{24} entre Ω_2 y Ω_4, Γ_{34}, entre Ω_3 y Ω_4. En cada frontera interna se consideran 501 grados de libertad (DOF) para que, al igual que los ejemplos anteriores la longitud de cada arista $\mathcal{S}'$ sea $h_{\mathcal{S}'} = 0.001$.

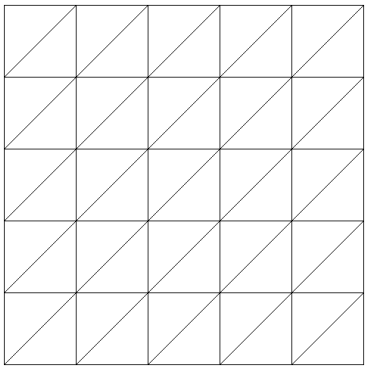


Figura 3.15: Malla inicial en $\Omega_i, i = 1, 2, 3, 4$

Por el análisis de la Tabla 3.4 se constata que el mallado final $\mathcal{T}_n$ en los subdominios Ω_1, Ω_2, Ω_3 y Ω_4 tiene respectivamente 56561, 25579, 231777 y 237320 grados de libertad (DOF). Obsérvese, que es en la primera iteración de Uzawa en la que se realiza un mayor número de adaptaciones de los mallados, en las iteraciones siguientes se hacen como mucho 3 adaptaciones. El mayor número de adaptaciones de mallado, 49, ocurre en Ω_1 pero aún así el número de grados de libertad es menor que en los subdominios Ω_3 y Ω_4, que son aquellos que al final tienen mayor número de grados de libertad. Este fenómeno ocurre debido a que la solución en estos subdominios es más suave y los errores en los elementos de la triangulación tienen el mismo orden de magnitud por lo que en cada adaptación se refina un gran número de elementos.

En las Figuras 3.16 y 3.17 se pueden ver para cada subdomio el error a-posteriori $\eta_{U_i}, i = 1, 2, 3, 4$ y el error global a-posteriori definido por $\eta_U := \eta_{U_1} + \eta_{U_2} + \eta_{U_3} + \eta_{U_4}$. Comparando el estimador del error obtenido con la recta de convergencia óptima, se observa un comportamiento quasi-óptimo en cada uno de los casos.

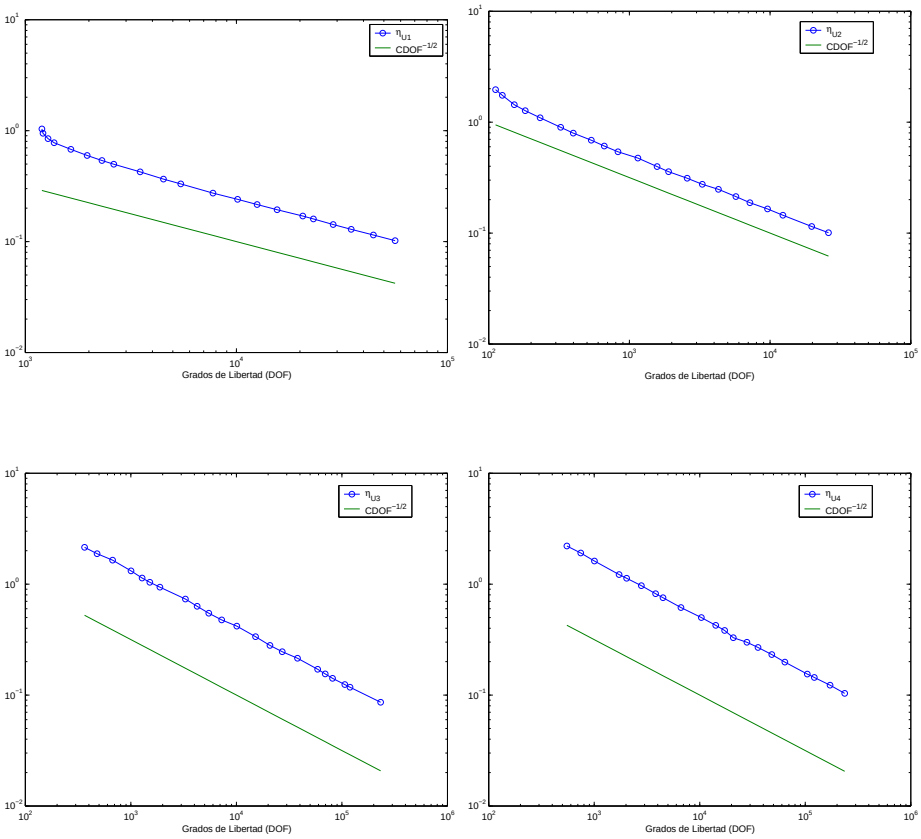


Figura 3.16: Error a-posteriori en Ω_i para $i = 1, 2, 3, 4$

En la Tabla 3.5 se presentan los errores obtenidos en los 4 multiplicadores de Lagrange y en la Figura 3.18 se pueden observar las gráficas que reflejan sus comportamientos. Al inicio, el error asociado al multiplicador en Γ_{12} es el que tiene mayor valor pero este decrece con el número de iteraciones. En cambio, los demás multiplicadores, presentan un error de Uzawa más pequeño, desde el inicio, decreciendo más lentamente y no se observan grandes alteraciones en su comportamiento.

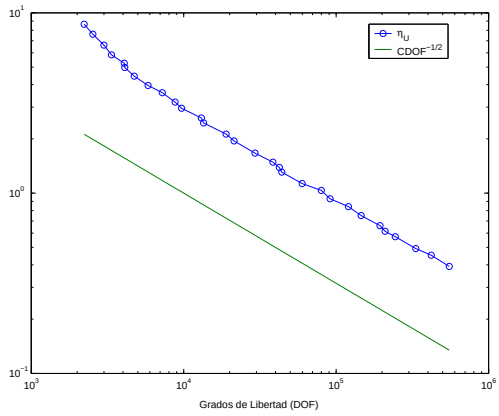


Figura 3.17: Error global a-posteriori

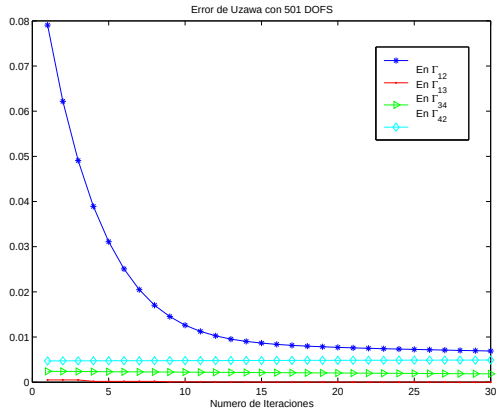


Figura 3.18: Error de Uzawa en las fronteras internas Γ_{ij}

Iter	R.A. Ω_1	R.A. Ω_2	R.A. Ω_3	R.A. Ω_4	DOF1	DOF2	DOF3	DOF4
1	16	4	6	9	1198	112	364	551
2	16	4	7	10	1198	112	479	746
3	16	5	8	11	1198	125	671	1004
4	16	6	9	11	1198	152	1004	1004
5	16	6	9	13	1198	152	1004	1722
6	16	7	9	13	1198	181	1004	1722
7	16	8	10	14	1198	231	1278	2027
8	16	8	11	15	1198	231	1512	2797
9	17	9	12	16	1204	326	1884	3816
10	19	10	14	16	1275	405	3289	3816
11	20	10	14	12	1332	405	3289	4498
12	21	11	15	18	1510	486	4244	6646
13	22	12	15	18	1720	680	4244	6646
14	23	13	17	20	2131	840	5480	10346
15	24	14	18	20	2618	1047	7265	10346
16	27	15	20	21	3559	1568	10092	14186
17	27	15	21	22	3559	1568	15183	17239
18	28	16	21	23	3559	1915	15183	23497
19	29	17	21	23	5870	2361	15183	23497
20	30	18	23	24	6778	3327	20794	28158
21	31	18	24	25	8025	3327	27222	35840
22	32	19	24	26	9494	3963	27222	48363
23	34	20	26	27	13441	5538	37511	64344
24	35	21	27	27	15665	7143	58681	64344
25	37	22	27	29	20198	8601	58681	104794
26	38	23	28	29	22852	12067	69342	104794
27	42	23	29	30	27945	12067	81486	119216
28	44	24	30	31	33731	14476	106329	172572
29	47	25	31	32	43201	17558	119419	237320
30	49	26	33	32	56561	25579	231777	237320

Tabla 3.4: Tabla de resultados para la descomposición en cuatro subdominios simétricos. Notación: `iter`, iteración del AMUADD; `R.A.`Ω_i, número de refinamientos acumulados en Ω_i; `DOFi`, número de grados de libertad en Ω_i.

Iter	$\|P_j - P_{j-1}\|_{\mathbb{M}_{12}}$	$\|P_j - P_{j-1}\|_{\mathbb{M}_{13}}$	$\|P_j - P_{j-1}\|_{\mathbb{M}_{34}}$	$\|P_j - P_{j-1}\|_{\mathbb{M}_{24}}$
1	7.90685e-2	4.92979e-4	2.41115e-3	4.70603e-3
2	6.21783e-2	4.96601e-4	2.38921e-3	4.71580e-3
3	4.90735e-2	4.97712e-4	2.36859e-3	4.72500e-3
4	3.89252e-2	1.78437e-4	2.34109e-3	4.73320e-3
5	3.11061e-2	1.78368e-4	2.31785e-3	4.74328e-3
6	2.50913e-2	1.78229e-4	2.29642e-3	4.75129e-3
7	2.04956e-2	1.82838e-4	2.28531e-3	4.74617e-3
8	1.70502e-2	1.82762e-4	2.26424e-3	4.75432e-3
9	1.45258e-2	6.39794e-5	2.24158e-3	4.76241e-3
10	1.26180e-2	6.43144e-5	2.22182e-3	4.77014e-3
11	1.12434e-2	6.42910e-5	2.20123e-3	4.77788e-3
12	1.02580e-2	2.20895e-5	2.18038e-3	4.78590e-3
13	9.53156e-3	2.20816e-5	2.16031e-3	4.79340e-3
14	9.03191e-3	2.24020e-5	2.14305e-3	4.79766e-3
15	8.66715e-3	7.36819e-6	2.12335e-3	4.80502e-3
16	8.37887e-3	7.40963e-6	2.10390e-3	4.81237e-3
17	8.15163e-3	2.25078e-6	2.08468e-3	4.81961e-3
18	7.98223e-3	2.25001e-6	2.06551e-3	4.82684e-3
19	7.83846e-3	2.24926e-6	2.04662e-3	4.83390e-3
20	7.71427e-3	2.27179e-6	2.02864e-3	4.84008e-3
21	7.60349e-3	6.18669e-7	2.01014e-3	4.84700e-3
22	7.50668e-3	6.18468e-7	1.99181e-3	4.85386e-3
23	7.41862e-3	6.21987e-7	1.97364e-3	4.86066e-3
24	7.33360e-3	1.94936e-7	1.95574e-3	4.86737e-3
25	7.25841e-3	1.94875e-7	1.93812e-03	4.87384e-3
26	7.18102e-3	1.96003e-7	1.92055e-03	4.88040e-3
27	7.10743e-3	1.96002e-7	1.90318e-03	4.8869e-3
28	7.03727e-3	3.32980e-22	1.8860e-3	4.89332e-3
29	6.96825e-3	3.40348e-22	1.86902e-3	4.89965e-3
30	6.90209e-3	2.73821e-22	1.85218e-3	4.90594e-3

Tabla 3.5: Tabla de resultados para la descomposición en cuatro subdominios simétricos. Notación: `iter`, iteración del AMUADD; $\|P_j - P_{j-1}\|_{\mathbb{M}_{ij}}$, error de Uzawa Γ_{ij}.

3.4.2. Ejemplo 4 ($h_{\mathcal{S}'} = 2.5e^{-4}$)

Para esta misma descomposición se presentan los resultados obtenidos en otra simulación, se consideran ahora mallados iniciales más finos, cada subdominio tiene 441 grados de libertad (DOF) (ver Figura 3.19) y en los mallados definidos en las fronteras internas, 2001 grados de libertad (DOF), con lo que se obtienen mallados en las que la longitud de cada arista $\mathcal{S}'$ es $h_{\mathcal{S}'} = 2.5e^{-4}$.

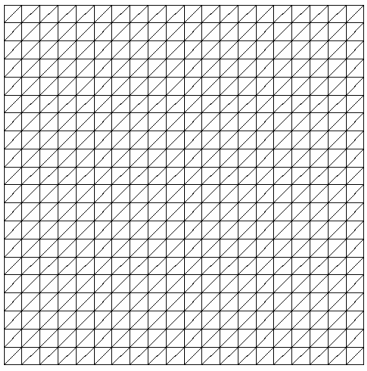


Figura 3.19: Malla inicial en $\Omega_i, i = 1, 2, 3, 4$

Se observa en la Tabla 3.6 que el mallado final $\mathcal{T}_n$ en los subdominios Ω_1, Ω_2, Ω_3 y Ω_4 tiene, respectivamente, 52763, 23734, 234828 y 272741 grados de libertad (DOF). Observar, que en general, el comportamiento del AMUADD es semejante al de la simulación anterior cuando se consideran mallas iniciales con 36 grados de libertad (DOF). En media, cada mallado se refina 9 veces menos que en el caso anterior y se sigue refinando más la malla de Ω_1 donde al final se ha adaptado la malla 39 veces, pero el número de grados de libertad es menor que en los subdominios Ω_3 y Ω_4. Esta simulación presenta un aumento significativo del número de grados de libertad en los subdominios Ω_3 y Ω_4 frente a la disminución

del número grados de libertad (DOF) en los mallados de Ω_1 y Ω_2.

En las Figuras 3.20 y 3.21 se presenta, para cada subdomio, el error a-posteriori $\eta_{U_i}, i = 1, 2, 3, 4$ y el error global a-posteriori, definido por $\eta_U := \eta_{U_1} + \eta_{U_2} + \eta_{U_3} + \eta_{U_4}$. Comparando el estimador del error obtenido con la recta de convergencia óptima, se observa un comportamiento quasi-óptimo en cada uno de los casos.

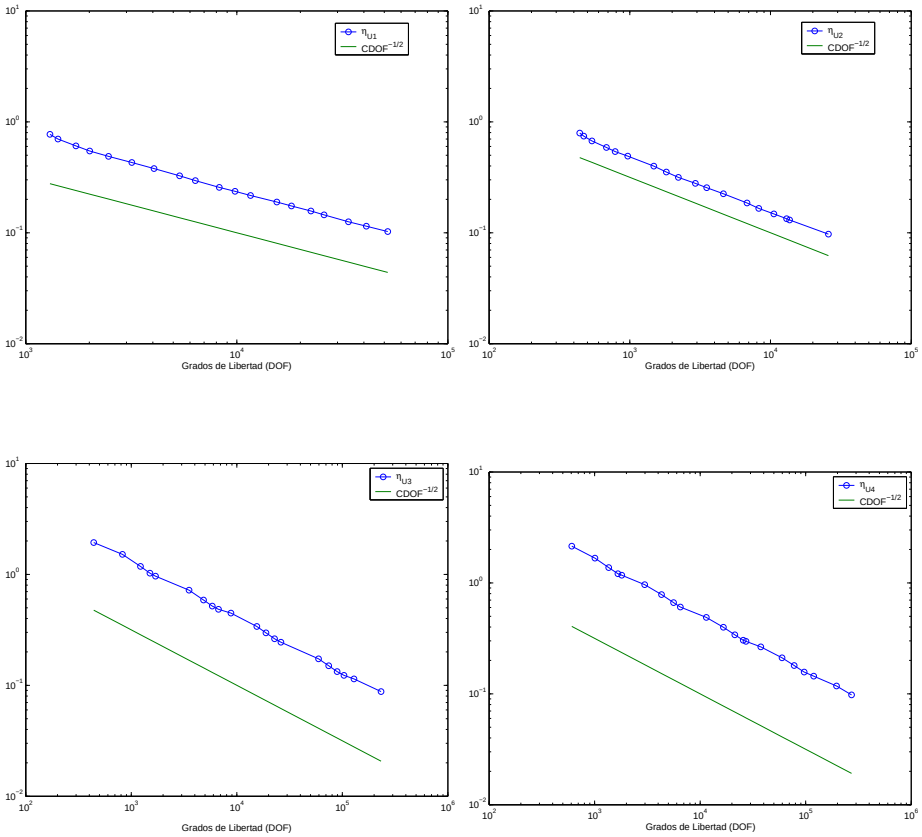


Figura 3.20: Error a-posteriori en Ω_i para $i = 1, 2, 3, 4$

Si comparamos estas gráficas con las de las Figuras 3.16 y 3.17, se observa que únicamente ha mejorado el error asociado a U_2 cuando se consideran mallados

iniciales más finos, sin embargo, en los demás subdominios el comportamiento de los errores es equivalente a cuando empezamos el cálculo de las soluciones con mallados más groseros.

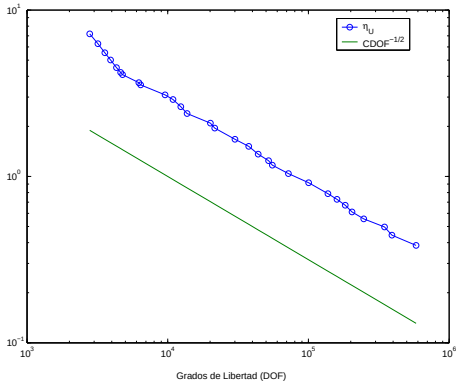


Figura 3.21: Error global a-posteriori

En la Tabla 3.7 se presentan los errores obtenidos en los 4 multiplicadores de Lagrange y en la Figura 3.22 se presentan las gráficas que reflejan sus comportamientos. Comparando las gráficas de esta figura con las de la Figura 3.18 se puede decir que los errores de Uzawa se comportan de la misma forma para 501 ó 2001 grados de libertad. Sin embargo, como era de esperar el hecho de tener más grados de libertad en el mallado nos permite obtener errores más pequeños. Obsérvese que cuando se elige un mallado en las fronteras internas aproximadamente 4 veces más fino que en el primer caso apenas conseguimos mejorar en un factor 2 la precisión en el cálculo de los multiplicadores.

En la Figura 3.23 presentamos las soluciones y mallados obtenidos en cada subdominio al final de 30 iteraciones.

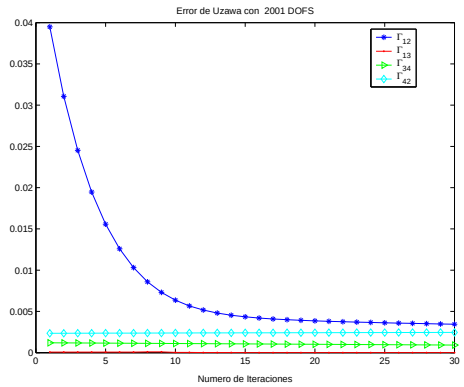


Figura 3.22: Error de Uzawa en las fronteras internas Γ_{ij}

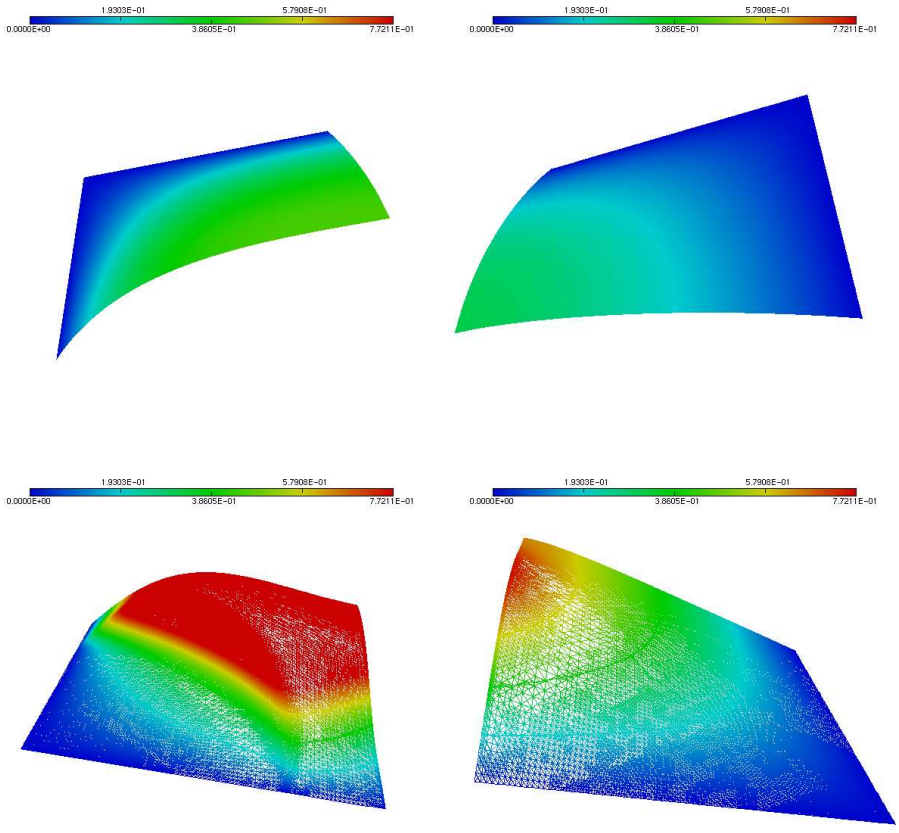


Figura 3.23: Solución en Ω_1 (abajo/izquierda), en Ω_2 (abajo/derecha), en Ω_3 (arriba/izquierda) y en Ω_4 (arriba/derecha)

Iter	R.A. Ω_1	R.A. Ω_2	R.A. Ω_3	R.A. Ω_4	DOF1	DOF2	DOF3	DOF4
1	7	0	0	1	1302	441	441	608
2	7	0	0	2	1302	441	441	1007
3	7	0	1	2	1302	441	825	1007
4	7	0	1	3	1302	441	825	1359
5	7	0	2	3	1302	441	1359	1359
6	7	0	2	4	1302	441	1359	1662
7	7	0	2	5	1302	441	1359	1809
8	7	0	3	6	1302	441	1503	2986
9	7	0	4	6	1302	441	1696	2986
10	7	0	6	7	1302	441	3521	4306
11	8	1	6	8	1328	471	3521	5596
12	10	2	7	8	1729	538	4839	5596
13	10	3	7	9	1729	682	4839	6492
14	11	4	9	11	2015	787	5858	11447
15	12	5	10	11	2454	974	6702	11447
16	14	6	12	12	3177	1497	8786	16604
17	15	6	13	12	4006	1497	15561	16604
18	16	7	13	13	5250	1836	15561	21291
19	16	8	14	14	5250	2245	18918	25627
20	18	9	14	15	6443	2949	18918	27216
21	19	10	15	16	8013	3561	22781	37458
22	21	11	16	17	9941	4845	26169	59673
23	23	11	19	17	12103	4845	59558	59673
24	25	12	19	18	14189	7014	59558	77744
25	27	13	19	19	19249	8356	59558	96687
26	28	14	20	19	22566	10818	74400	96687
27	31	15	21	21	27868	12943	89306	119216
28	33	16	22	22	32051	15606	103583	197762
29	37	17	24	22	43437	23734	130104	197762
30	39	17	25	23	52763	23734	234828	272741

Tabla 3.6: Tabla de resultados para la descomposición en cuatro subdominios simétricos. Notación: `iter`, iteración del AMUADD; `R.A.`Ω_i, número de refinamientos acumulados en Ω_i; `DOFi`, número de grados de libertad en Ω_i.

Iter	$\|P_j - P_{j-1}\|_{\mathbb{M}_{12}}$	$\|P_j - P_{j-1}\|_{\mathbb{M}_{13}}$	$\|P_j - P_{j-1}\|_{\mathbb{M}_{34}}$	$\|P_j - P_{j-1}\|_{\mathbb{M}_{24}}$
1	3.94972e-2	7.19916e-5	1.20031e-3	2.35436e-3
2	3.10622e-2	7.19641e-5	1.19225e-3	2.35413e-3
3	2.45249e-2	7.19461e-5	1.18266e-3	2.35833e-3
4	1.94699e-2	7.1919e-5	1.17212e-3	2.36202e-3
5	1.55741e-2	7.24029e-05	1.16293e-03	2.36614e-3
6	1.25865e-2	7.23758e-5	1.15266e-03	2.36975e-3
7	1.031061e-2	7.23491e-5	1.14185e-03	2.37385e-03
8	8.59224e-3	9.75391e-5	1.13205e-03	2.37867e-03
9	7.30833e-3	9.76607e-5	1.12177e-03	2.38264e-3
10	6.35976e-3	3.33115e-5	1.111058e-03	2.38673e-3
11	5.66669e-3	3.32995e-5	1.10022e-03	2.39065e-3
12	5.16715e-3	1.16835e-5	1.08999e-03	2.39443e-3
13	4.80005e-3	1.16793e-5	1.07991e-03	2.39827e-3
14	4.53585e-3	1.17465e-5	1.07121e-03	2.40052e-3
15	4.34839e-3	4.08523e-6	1.06129e-03	2.40425e-3
16	4.20156e-3	4.1125e-6	1.05145e-03	2.40799e-3
17	4.08653e-3	1.40225e-6	1.04183e-03	2.41164e-3
18	3.99542e-3	1.40178e-6	1.03222e-03	2.41527e-3
19	3.91852e-3	1.40988e-6	1.02303e-03	2.4185e-3
20	3.85519e-3	1.4094e-6	1.01362e-03	2.42205e-3
21	3.80114e-3	1.40904e-6	1.00434e-03	2.42905e-3
22	3.75083e-3	4.63375e-7	9.95123e-4	2.42905e-3
23	3.70453e-3	1.41326e-7	9.86032e-4	2.43249e-3
24	3.66226e-3	1.41281e-7	9.77006e-04	2.4359e-3
25	3.6230e-3	1.41237e-7	9.68149e-04	2.43918e-3
26	3.58495e-3	1.42052e-7	9.59323e-04	2.44252e-3
27	3.54865e-3	1.42012e-7	9.50589e-4	2.44582e-3
28	3.51341e-3	3.87049e-8	9.4194e-04	2.44909e-3
29	3.47932e-3	3.89302e-8	9.33882e-04	2.45232e-3
30	3.44579e-3	1.21718e-8	9.2493e-04	2.4555e-3

Tabla 3.7: Tabla de resultados para la descomposición en cuatro subdominios simétricos. Notación: `iter`, iteración del AMUADD; $\|P_j - P_{j-1}\|_{\mathbb{M}_{ij}}$, error de Uzawa Γ_{ij}.

3.5. Ejemplo 5

En este ejemplo se considera, como en el Ejemplo 3.4, una descomposición del dominio inicial Ω en cuatro subdominios, pero ahora $\Omega_1 = [0, 0.3] \times [0, 0.3]$, $\Omega_2 = [0.3, 1] \times [0, 0.3]$, $\Omega_3 = [0, 0.3] \times [0.3, 1]$ y $\Omega_4 = [0.3, 1] \times [0.3, 1]$. Con esta descomposición se definen 4 subdominios con diferentes dimensiones (ver Figura 3.24). En los subdominios Ω_1, Ω_2, Ω_3 y Ω_4 se considera, un mallado inicial con respectivamente 16, 32, 32 y 64 grados de libertad (DOF) y cuatro multiplicadores de Lagrange definidos sobre las fronteras internas, Γ_{12} entre Ω_1 y Ω_2, Γ_{13} entre Ω_1 y Ω_3, Γ_{24} entre Ω_2 y Ω_4, Γ_{34} entre Ω_3 y Ω_4. En las fronteras internas Γ_{12} y Γ_{13} consideramos mallas unidimensionales con 1001 grados de libertad (DOF), quedando cada arista $\mathcal{S}'$ con $h_{\mathcal{S}'} = 3e^{-4}$. En las fronteras internas Γ_{13} y Γ_{42} consideramos mallas unidimensionales con 2001 grados de libertad (DOF), quedando cada arista $\mathcal{S}'$ con $h_{\mathcal{S}'} = 3.5e^{-4}$.

En la Figura 3.25 presentamos las soluciones y mallas obtenidas en cada subdominio al final de 30 iteraciones. Por el análisis de la Tabla 3.8 se constata que el mallado final $\mathcal{T}_n$ en los subdominios Ω_1, Ω_2, Ω_3 y Ω_4 tiene respectivamente 43029, 37586, 122251 y 145703 grados de libertad (DOF). Además, se observa que es en las primeras iteraciones en donde se hacen más adaptaciones. Sin embargo, en las siguientes iteraciones, se hacen como mucho 2 adaptaciones en cada subdominio, por cada iteración de Uzawa. El mayor número de adaptaciones de mallado, 37, ocurre en Ω_3.

Los subdominios que al final tienen mayor número de grados de libertad, tal como en el ejemplo anterior, son Ω_3 y Ω_4. Pero con esta descomposición el número de grados de libertad en esos dominios es significativamente menor, tienen como mucho la mitad de los grados de libertad que en el ejemplo anterior.

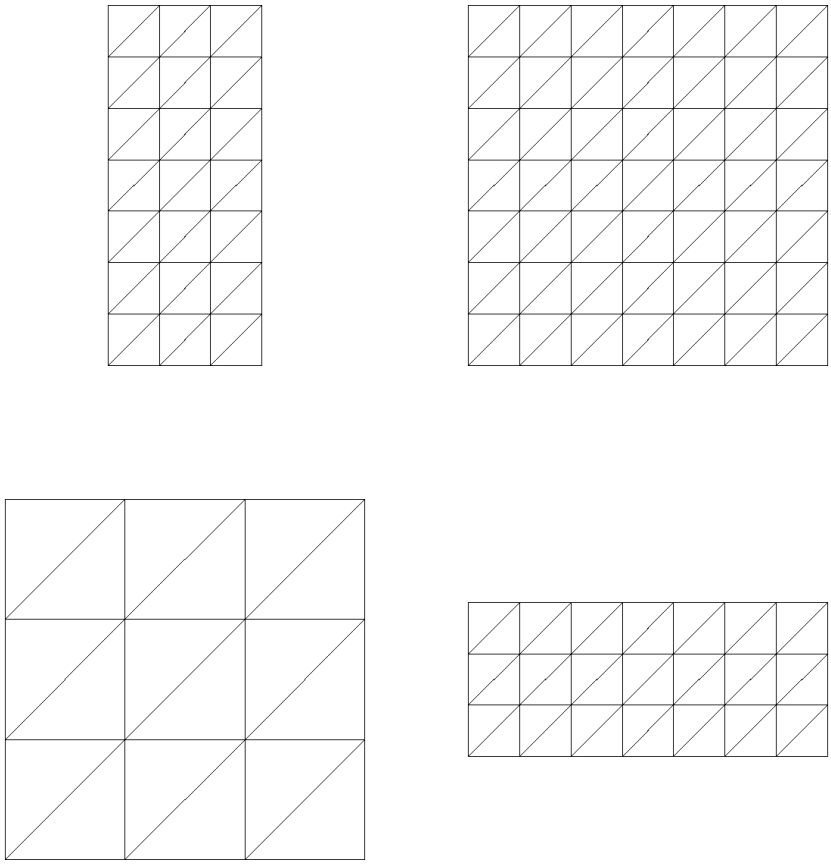


Figura 3.24: Mallas iniciales en Ω_1 (abajo/izquierda), en Ω_2 (abajo/derecha), en Ω_3 (arriba/izquierda) y en Ω_4 (arriba/derecha)

En la Figura 3.27 se presentan las gráficas que reflejan el comportamiento del error de Uzawa en cada uno de los cuatro multiplicadores de Lagrange y en la Tabla 3.9 los valores de los referidos errores. Se observa que todos los multiplicadores presentan errores del orden de 10^{-3} y que el error asociado al multiplicador definido sobre la frontera Γ_{42} es más grande que en los demás multiplicadores.

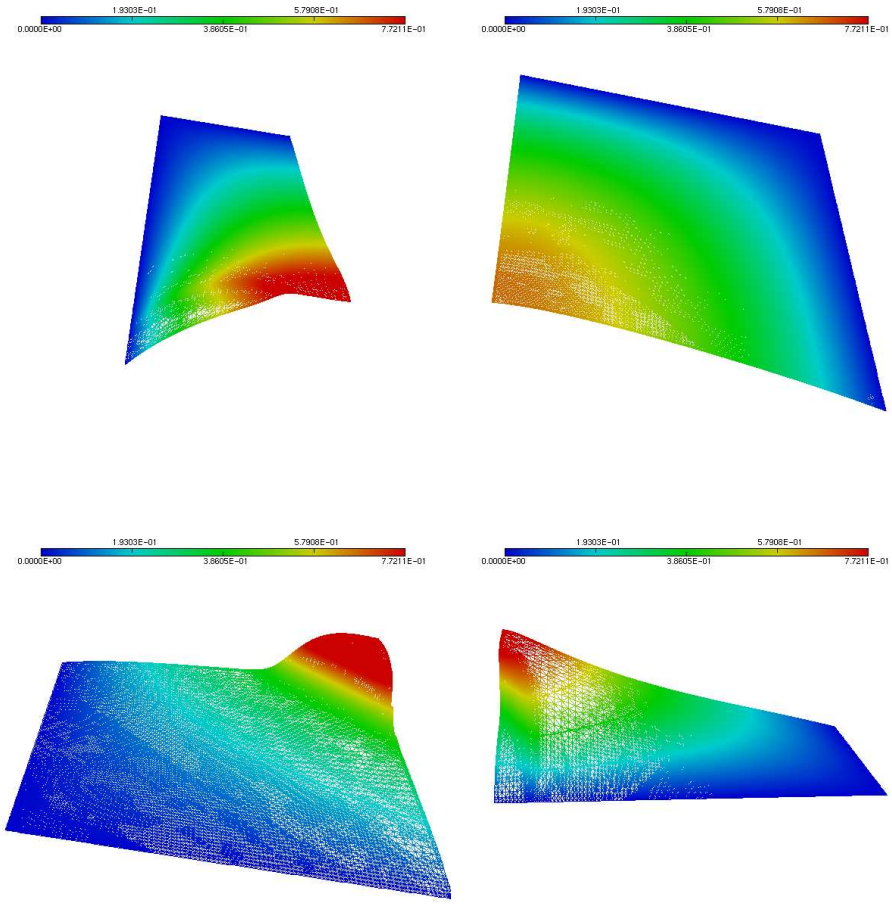


Figura 3.25: Solución en Ω_1 (abajo/izquierda), en Ω_2 (abajo/derecha), en Ω_3 (arriba/izquierda) y en Ω_4 (arriba/derecha)

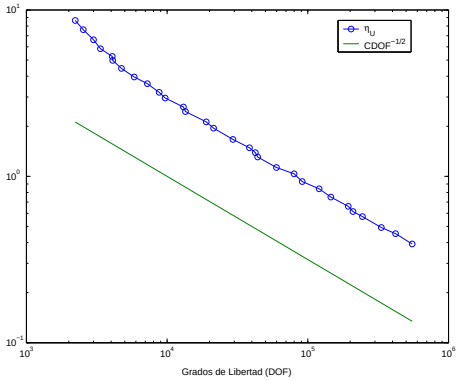


Figura 3.26: Error global a-posteriori

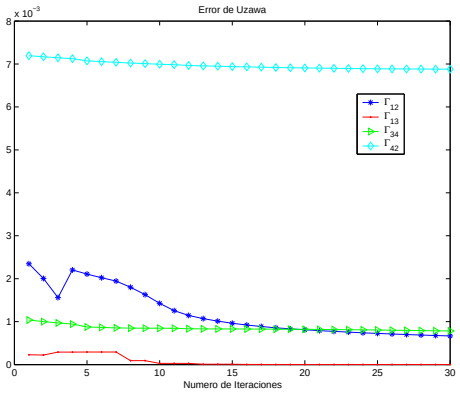


Figura 3.27: Error de Uzawa en las fronteras internas Γ_{ij}

Iter	R.A.Ω_1	R.A.Ω_2	R.A.Ω_3	R.A.Ω_4	DOF1	DOF2	DOF3	DOF4
1	0	0	2	2	49	105	246	411
2	0	0	2	2	49	105	246	411
3	1	2	4	3	57	145	335	528
4	8	2	5	3	269	145	562	528
5	8	3	5	5	269	188	562	753
6	9	3	6	6	340	188	793	1145
7	10	4	6	6	421	232	793	1145
8	11	5	7	7	589	285	930	1640
9	11	6	8	7	589	391	1089	1640
10	12	7	10	8	813	484	1372	2062
11	12	8	11	9	813	634	2200	2400
12	13	9	11	10	956	857	2200	2805
13	14	9	12	11	1209	857	3011	3886
14	15	10	13	12	1568	1061	3161	5928
15	16	13	15	12	2107	1572	3785	5928
16	17	14	16	13	2646	2021	4738	7610
17	18	15	18	14	3396	2739	6302	9151
18	18	15	19	15	3396	2739	10776	10545
19	19	16	19	16	4068	3583	10776	13015
20	20	17	20	17	5318	4406	11547	20348
21	21	19	23	17	6779	6025	13963	20348
22	22	19	25	18	8150	6025	20350	27084
23	23	21	27	19	10790	8756	24786	32804
24	24	22	28	20	13500	11440	32864	38458
25	25	23	29	21	16633	12368	38458	45370
26	26	24	30	22	21067	14935	43570	65340
27	27	26	32	23	26830	18576	55784	96959
28	28	27	33	23	32832	23718	59929	96959
29	29	29	35	24	36887	28345	84168	122251
30	30	31	37	25	43029	37586	122251	145703

Tabla 3.8: Tabla de resultados para la descomposición en cuatro subdominios no simétricos. Notación: `iter`, iteración del AMUADD; R.A.Ω_i, número de refinamientos acumulados en Ω_i; `DOFi`, número de grados de libertad en Ω_i.

$Uzawa$	$\|P_j - P_{j-1}\|_{\mathbb{M}_{12}}$	$\|P_j - P_{j-1}\|_{\mathbb{M}_{13}}$	$\|P_j - P_{j-1}\|_{\mathbb{M}_{34}}$	$\|P_j - P_{j-1}\|_{\mathbb{M}_{24}}$
1	2.34722e-3	2.28665e-4	1.03862e-3	7.19375e-03
2	2.00448e-3	2.229054e-4	9.999703e-04	7.16822e-03
3	1.55997e-3	2.90771e-4	9.69252e-04	7.14637e-03
4	2.20359e-3	2.91216e-4	9.43822e-04	7.12483e-03
5	2.10667e-3	2.91563e-4	8.76112e-04	7.07272e-03
6	2.02249e-3	2.93047e-4	8.65381e-04	7.05455e-03
7	1.94547e-3	2.9332e-4	8.55183e-04	7.0386e-03
8	1.80056e-3	9.18422e-5	8.4932e-04	7.02258e-03
9	1.62823e-3	9.12645e-5	8.47042e-04	7.00896e-03
10	1.4252e-3	2.72217e-5	8.47299e-04	6.99366e-3
11	1.25498e-3	2.7241e-5	8.45194e-04	6.98376e-3
12	1.14465e-3	2.7256e-5	8.33561e-04	6.96538e-3
13	1.06992e-3	9.12978e-6	8.32383e-04	6.95597e-3
14	1.01219e-3	9.14142e-6	8.3053e-04	6.94772e-3
15	9.62044e-4	9.11654e-6	8.29604e-04	6.93944e-3
16	9.19994e-4	3.03971e-6	8.28051e-04	6.93271e-3
17	8.87459e-4	3.05735e-6	8.26813e-04	6.92608e-3
18	8.57299e-4	9.80447e-7	8.24116e-04	6.91838e-3
19	8.31477e-4	9.80702e-7	8.22166e-04	6.9129e-3
20	8.11488e-4	9.86469e-7	8.19956e-04	6.908e-3
21	7.90739e-4	9.86556e-7	8.17611e-04	6.90346e-3
22	7.7258e-4	2.8451e-7	8.14943e-04	6.8994e-3
23	7.55553e-4	2.86403e-7	8.11933e-04	6.8957e-3
24	7.40004e-4	2.864e-7	8.08278e-04	6.89191e-3
25	7.25545e-4	8.53141e-8	8.0456e-04	6.88891e-3
26	7.12072e-4	8.58668e-8	8.00563e-04	6.88621e-3
27	6.9999e-4	8.5886e-8	7.96317e-4	6.88379e-3
28	6.88125e-4	8.5896e-8	7.91798e-4	6.88162e-3
29	6.77186e-4	2.2291e-8	7.8696e-4	6.8796e-3
30	6.66993e-4	2.22993e-8	7.81913e-4	6.8779e-3

Tabla 3.9: Tabla de resultados para la descomposición en cuatro subdominios no simétricos. Notación:`iter`, iteración del AMUADD; $\|P_j - P_{j-1}\|_{\mathbb{M}_{ij}}$, error de Uzawa Γ_{ij}.

Respecto a los ejemplos presentados se hacen a continuación algunas reflexiones generales sobre los resultados obtenidos. Obsérvese, que el AMUADD nos permite resolver el Problema 3.1 en cualquiera de las descomposiciones consideradas, mientras que usando el algoritmo de Uzawa modificado sin adaptación de mallado, hay descomposiciones para las cuales no es posible solucionar el problema, aunque se consideren mallados muy finos. Con el AMUADD, se obtiene mayor precisión en las aproximaciones U_i de las soluciones y en los multiplicadores de Lagrange, así como una mayor rapidez de convergencia frente a la resolución del problema usando el algoritmo de Uzawa modificado sin adaptación de mallado. En relación a la convergencia de nuestro algoritmo, comparando las gráficas de los indicadores de error con las rectas de convergencia óptima, se observa un comportamiento quasi-óptimo en cada uno de los casos.

Una consecuencia de la resolución del problema de forma independiente en cada subdominio, está en refinar más los mallados correspondientes a subdominios donde la solución es más suave, por el hecho que el error presentado por cada elemento de la triangulación es del mismo orden de magnitud. Esto podría parecer un inconveniente, sin embargo, los órdenes de convergencia son óptimos. Además, el AMUADD captura muy bien las capas límites, incluso en el caso en que la frontera entre dos subdominios dominios cruce la capa límite.

Capítulo 4

Aspectos de Programación

El objetivo de este capítulo es comentar algunos detalles de la programación del AMUADD, en particular los algoritmos de refinamiento para las mallas, en una y dos dimensiones y como manejamos funciones del tipo elemento finito, como funciones definidas por una expresión aritmética.

4.1. Refinamiento de elementos marcados

Antes de comentar el algoritmo, se detalla la estrategia de marcado que se ha utilizado. Esta estrategia selecciona qué elementos de la triangulación se deben refinar para que el procedimiento genere una sucesión de soluciones discretas. La estrategia de marcado que hemos utilizado es la del máximo que garantiza la reducción de error y se resume en la tabla:

Estrategia de Marcado

Dado un parámetro $0 < \gamma < 1$:

1. Seleccionar un conjunto $\widehat{T_H}$ de $\mathcal{T}_H$ tal que,

$$\eta_{\hat{T}} > \gamma \max \eta_{\hat{T}}, \text{ para todo } \hat{T} \in \mathcal{T}_H.$$

2. Marcar los elementos de $\widehat{T_H}$.

Es decir, se utiliza un refinamiento por indicador en los mallados de cada subdominio donde los elementos marcados de $\widehat{T_H}$ se refinan usando el refinamiento $4T$ de Rivara [47, 48]. Este refinamiento (ver Figura 4.1) consiste en subdividir un triángulo dado $\hat{T} \in \widehat{T_H}$, primero en dos triángulos buscando el lado mayor de $\hat{T}$ (en rojo) en el cual se crea un nodo interior en el punto medio de la arista y enseguida se subdividen cada uno de los dos triángulos obtenidos, buscando el lado mayor en cada uno de ellos donde se introduce un nodo interior. De este modo, el elemento de la triangulación $\hat{T} \in \mathcal{T}_H$ queda refinado, dando origen a 4 elementos en la triangulación $\mathcal{T}_{h_1} \subset \mathcal{T}_H$. Muchas veces, después de refinar el elemento $\hat{T} \in \widehat{T_H}$ surge la necesidad de hacer refinamientos en elementos adyacentes para restablecer la continuidad (ver Figuras 4.2 y 4.3), buscando también el lado mayor del triángulo para crear el nodo interior (en rojo en las Figuras 4.2 y 4.3).

Para el desarrollo de los experimentos numéricos se ha usado el programa de elementos finitos neptuno++ de L. Ferragut [21], al que se han añadido importantes módulos que pasamos a exponer con más detalle y al que llamamos neptunoDD++.

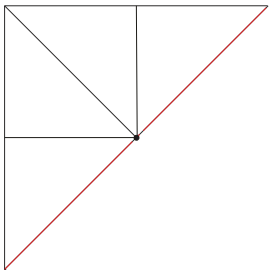


Figura 4.1: Refinamiento de un elemento $\hat{T}$

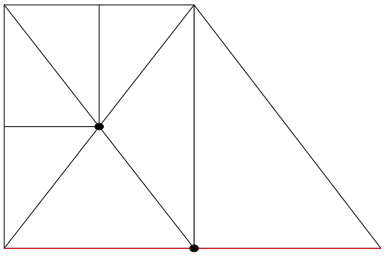


Figura 4.2: Refinamiento de un elemento adyacente a $\hat{T}$

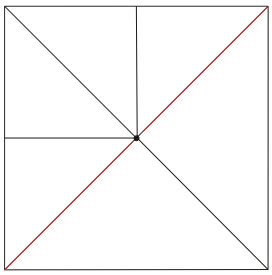


Figura 4.3: Refinamiento de un elemento adyacente a $\hat{T}$

Los principales módulos que ha supuesto este trabajo son:

- Incorporación de la clase $mesh1d.h$, para definir una malla en la frontera Γ_{lk}, constituida por las siguientes funciones:

  - Funciones de construcción y destrucción de mallas así como también otras funciones necesarias para la manipulación de mallas:

```
// constructor
mesh1d (int NUMNP=0, int NUMEL=0, int NDM=2,  int NEL=2);

// constructor  de una malla rectangular
mesh1d (const int nrow, const double hy, const double *xorig,
       const int nr1=1, const int nr2=2, const int nr3=3,
       const int nr4=4);

// constructor de copia
 mesh1d (const mesh1d& orig);

// destructor
~mesh1d ();

// operador de asignacion
mesh1d& operator=(const mesh1d& orig);

// genera la nueva malla
void gmesh (mesh1d& O);
```

- Establecimiento del código de refinamiento y creación de funciones para refinar una malla según el indicador de error:

```
// Establece el codigo de refinamiento
void refcod (const vect_d &ErrInd);

// refina la malla
void refine (const vect_d& ErrInd);

// refina una malla
void refine(mesh1d& O, const vect_d& ErrInd);

\\ Construye la tabla de conexiones nodales
void build_ix();
```

- Función para la creación de un fichero para dibujar con MEDIT ([22]).

```
void plot();
```

- Función que retorna el número del elemento al que pertenece el punto de coordenadas xcoor.

```
int get_element(double xcoor[]);
```

- Incorporación de la clase $fe\_field1d.h$ (subclase de la clase fe_field) constituida por funciones para operar con elementos finitos P^1. Destacamos la siguiente función que nos devuelve el valor del $fe\_field1d$ en el punto xcoor.

```
// valor de la funcion en el punto de coordenadas xcoor
   double operator() (double *xcoor) const;
```

- Incorporación de la clase $problema1d.h$, constituida por todas la funciones necesarias para la resolución de los problemas sobre las fronteras internas Γ_{lk}. Entre otras, se han programado funciones para la construcción de la matriz de masa del sistema y para el cálculo del segundo miembro con los correspondientes ensamblajes, etc....

- Programación de las funciones para localizar a que elemento del mallado definido en Ω_i pertenece un punto dado x y para el cálculo del valor de una función de tipo elemento finito P^1 (fe_field) en un punto dado de coordenadas xcoor.

A continuación se comentan algunos detalles sobre la implementación del refinamiento.

Como se ha comentado anteriormente, la adaptación del mallado se hace refinando en cada subdominio los elementos marcados en los mallados correspondientes, de modo que en cada arista se crea un nodo interior, es decir, cada elemento triangular se refina originando 4 nuevos triángulos a los que llamamos hijos del elemento refinado. Con este tipo de refinamiento, al final se obtiene un conjunto

de mallas encajadas en cada uno de los subdominios Ω_i considerados en la descomposición. A continuación se designa por $\mathcal{T}_H$ el mallado inicial y por $\mathcal{T}_{hi}$ el mallado obtenido en la i-ésima adaptación, es decir, tenemos:

$$\mathcal{T}_H \supset \mathcal{T}_{h1} \supset \mathcal{T}_{h2} \ldots \supset \mathcal{T}_{hi} \ldots \supset \mathcal{T}_{hmin} \tag{4.1}$$

donde $\mathcal{T}_{hmin}$ es la malla más fina. El hecho de que las mallas estén encajadas es de gran importancia a la hora de localizar un punto de coordenadas *xcoor* en un elemento del mallado más fino. Cada vez que se adapta un mallado, en un subdominio dado, se construye una tabla con los hijos de cada elemento de la triangulación que se ha refinado y con la ayuda de dos punteros se guarda la información de quién es la malla padre y de qué malla son hijos dichos elementos. Así, a la hora de localizar un punto dado de coordenadas *xcoor* en la malla fina $\mathcal{T}_{hmin}$, empezamos por localizar ese punto en un elemento T de la malla inicial $\mathcal{T}_H$ (la más gruesa y por lo tanto aquella que está constituida por un número menor de elementos). A continuación, se localiza ese punto en un elemento T_1 de la malla $\mathcal{T}_{h1}$ haciendo la búsqueda únicamente en sus hijos, resultantes del refinamiento de T en la malla $\mathcal{T}_H$, y no en todos los elementos de la malla $\mathcal{T}_{h1}$. A la hora de localizar dicho punto en un elemento de $\mathcal{T}_{h2}$, es suficiente hacer la búsqueda en los hijos de T_1 que están en $\mathcal{T}_{h2}$, y así sucesivamente hasta que que al final se localiza el punto de coordenada *xcoor* en un elemento de $\mathcal{T}_{hmin}$, el mallado más fino. Esta búsqueda es muy eficiente y se puede mirar como una búsqueda en profundidad, donde cada nivel es un mallado más fino que el mallado del nivel anterior. Si no utilizáramos este algoritmo de búsqueda, a la hora de localizar el punto en un elemento del último nivel (malla más fina) el número de elementos donde habría que buscar aumentaría significativamente, por lo que el algoritmo no sería eficiente.

En el caso de una malla con una frontera interna Γ_{lk}, cada elemento del mallado es un intervalo real y el refinamiento de un elemento dado se hace por división del intervalo en dos partes iguales. El algoritmo de búsqueda que permite localizar un punto en un elemento de un mallado, funciona de la misma forma que el que acabamos de exponer para las mallas definidas en los subdominios Ω_i, se empieza por localizar el punto en la malla más gruesa y después se hace la búsqueda en sus hijos y así sucesivamente hasta llegar a localizar el punto en un elemento de la malla más fina.

4.2. Cálculo del valor de una función de tipo elemento finito

Se calcula el valor de una función u de tipo elemento finito P^1 en un punto dado x de coordenadas $xcoor$, de la siguiente forma :

- Empezamos por localizar el punto en un elemento de la triangulación del mallado usando la función,

  ```
  int get_element(doublexcoor[]);
  ```

  que devuelve el número del elemento.

- Calculamos las coordenadas baricéntricas $(\lambda_0, \lambda_1, \lambda_2)$ del punto de coordenadas $xcoor$.

- Calculamos el valor de la función u del tipo elemento finito en el punto $xcoor$ como resultado de la combinación lineal de los valores de la función u en los vértices del triángulo (nodos del elemento), numerados con $0, 1, 2$,

es decir,

$$u[coor] = \lambda_0 * u[0] + \lambda_1 * u[1] + \lambda_2 * u[2] \tag{4.2}$$

Para hacer el cálculo del valor de la función u de tipo elemento finito en un punto dado de coordenadas $xcoor$ en Γ_{lk}, el procedimiento es igual al descrito anteriormente. Se localiza el punto en un elemento dado de la malla sobre la frontera interna Γ_{lk}, después se calculan las coordenadas baricéntricas (λ_0, λ_1) de ese punto y el valor de u en el punto de coordenadas $xcoor$ viene dado por la siguiente expresión,

$$u[coor] = \lambda_0 * u[0] + \lambda_1 * u[1] \tag{4.3}$$

Capítulo 5

Aplicación del AMUADD a Problemas de Convección-Reacción-Difusión

5.1. Introducción

El objetivo de este capítulo es extender la aplicación de nuestro algoritmo AMUADD a la resolución de problemas de convección-reacción-difusión.
El AMUADD es un algoritmo potente ya que también funciona sin mallas encajadas y con otros tipos de adaptadores de mallado. El ejemplo que presentamos está desarrollado con Freefem++ [29], que es un ambiente de desarrollo integrado de alto nivel para resolver numéricamente problemas de ecuaciones en derivadas parciales y cuyas bibliotecas de rutinas están implementadas en el lenguaje $C++$. El Freefem tiene un adaptador de mallado anisótropo. A pesar de no tener mallas encajadas tiene una rutina llamada "Quatri"que le permite determinar en qué elemento del mallado está un punto dado de coordenadas (x, y). Como hemos visto anteriormente, el algoritmo AMUADD converge a la solución deseada cuando

consideramos un espacio de dimensión finita que corresponda a un mallado suficientemente fino. Vamos, ahora a generalizar nuestro algoritmo a problemas no lineales. En este caso los espacios no están encajados, sin embargo, en el Teorema 2.2 sí, pero eso no nos supone ningún problema ya que la cuestión principal para la convergencia es el hecho de que en el problema auxiliar (2.28) el error ε_j va disminuyendo en cada paso y, en nuestro experimento, aunque los $\mathbb{V}_j$ no estén estrictamente contenidos en $\mathbb{V}_{j+1}$, $\mathbb{V}_{j+1}$ es una mejor aproximación de $\mathbb{V}_J$. En estas adaptaciones, también se desrefinan los mallados porque la solución calculada en cada paso va a tener que ser utilizada en el paso siguiente en el término difusivo, donde la incógnita u aparece de forma explícita. A la hora de desrefinar, se aproxima la solución en el mallado más grueso utilizando el error de interpolación.

5.2. Resultados Numéricos

En esta sección, presentamos los resultados numéricos que hemos obtenido con la aplicación de nuestro algoritmo AMUADD a la resolución de un problema estacionario no lineal.

Consideremos 2 subdominios, un rectángulo menos un triángulo unitario $\Omega_2 = \{(x, y) : 0 < x < 2, 0 < y < 1, x + y > 1\}$ y un círculo unitario Ω_1 centrado en la esquina izquierda inferior del rectángulo. Este problema presenta distintas características en cada uno de los subdominios: en Ω_1 tenemos un problema fuertemente convectivo y en Ω_2 un problema no lineal, con condiciones de contorno

Dirichlet no homogéneas. El problema está definido por las siguientes ecuaciones:

$$-\Delta u + \overrightarrow{v}\nabla u = 1 \quad \texttt{en } \Omega_1 \tag{5.1}$$

$$-\Delta u = e^u \quad \texttt{en } \Omega_2 \tag{5.2}$$

$$u = 1 \quad \texttt{en } \Gamma_i \quad i = 1, 2 \tag{5.3}$$

con $\overrightarrow{v} = (100x, -100y)$.

A continuación, presentamos en la Figura 5.1 los mallados iniciales considerados en cada uno de los subdominios (arriba) y los mallados finales obtenidos (abajo). Se observa que en Ω_1 hay regiones donde claramente se ha desrefinado el mallado, es decir, sitios donde el mallado correspondiente a la solución final es más grueso que el mallado inicialmente considerado.

En cuanto a los indicadores de error η_{U_1} y η_{U_2} de las soluciones U_1 y U_2 se observa en la Figura 5.3 que en Ω_2 la recta de convergencia tiene un comportamiento quasi-óptimo. Sin embargo, no podemos decir lo mismo en relación al error en U_1, ya que este tiene un comportamiento un poco oscilatorio, pero aún así los valores de los errores están por debajo de los valores presentados por la recta de convergencia óptima. Se define por $\eta_U := \eta_{U_1} + \eta_{U_2}$ el indicador de error en la solución global $U = U_1 + U_2$. Comparando el estimador del error obtenido con la recta de convergencia óptima, se observa un comportamiento quasi-óptimo en el cálculo de la solución global U.

A continuación, presentamos en la Figura 5.2 (arriba) las soluciones U_1 y U_2 por separado y hacemos la recomposición de las soluciones y mallas finales obtenidas en cada subdominio (abajo). Se observa que hay continuidad de la solución en la frontera interna Γ_{12}.

El la Figura 5.4 se presenta la gráfica del comportamiento del error de Uzawa.

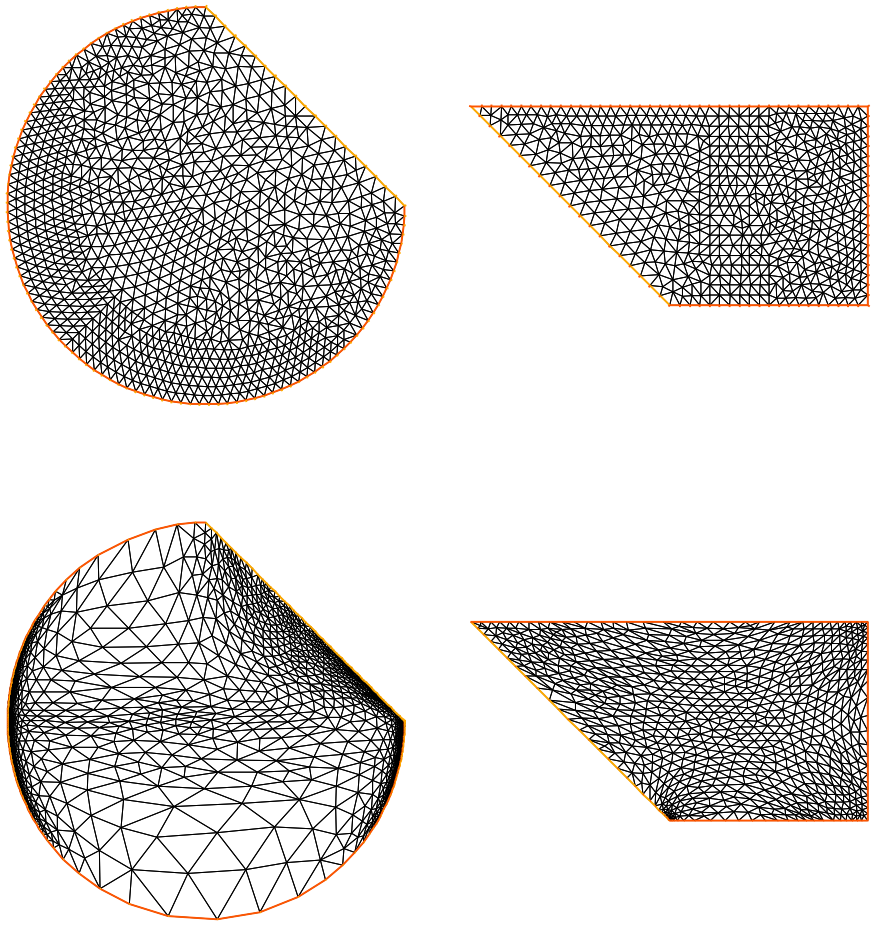


Figura 5.1: Los dos dibujos de arriba representan el mallado inicial considerado en cada subdominio y abajo los mallados finales.

Se observa como el error disminuye con el número de iteraciones, es decir, se tiene la convergencia en el multiplicador.

En la Tabla 5.1 presentamos un resumen de los resultados obtenidos en cada iteración, *iter*, los grados de libertad (DOF) en cada subdominio y el error de Uzawa.

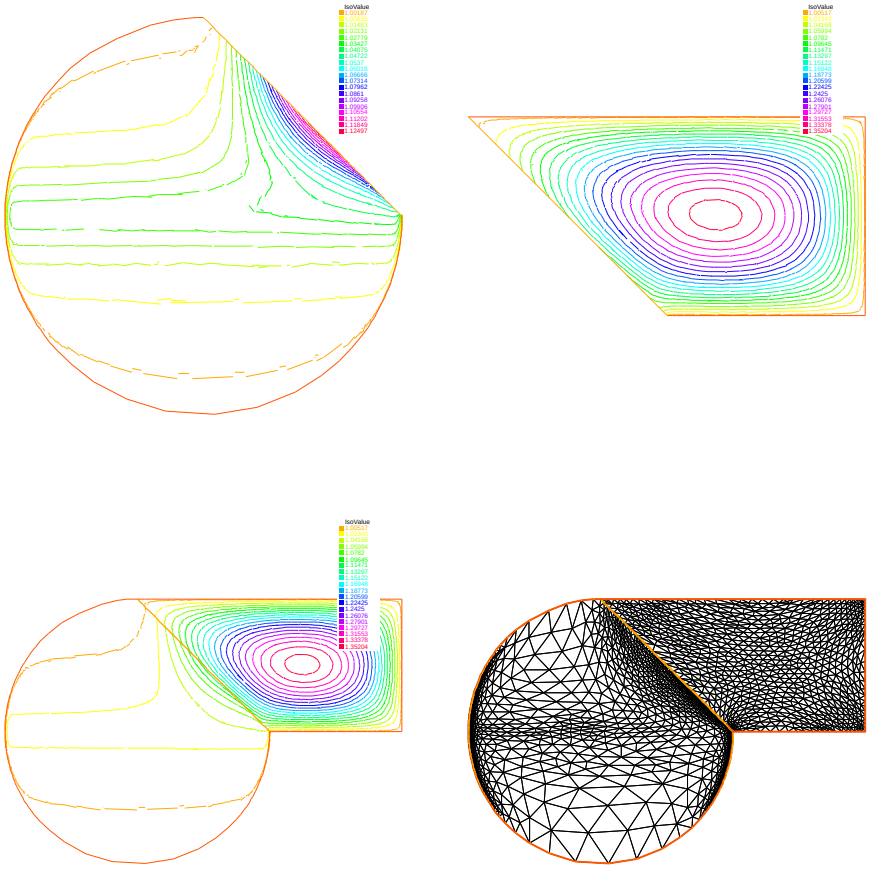


Figura 5.2: Los dos dibujos de arriba representan las soluciones descompuestas, abajo a la izquierda la recomposición de los dos dibujos de arriba y, abajo a la derecha, la recomposición de los mallados finales.

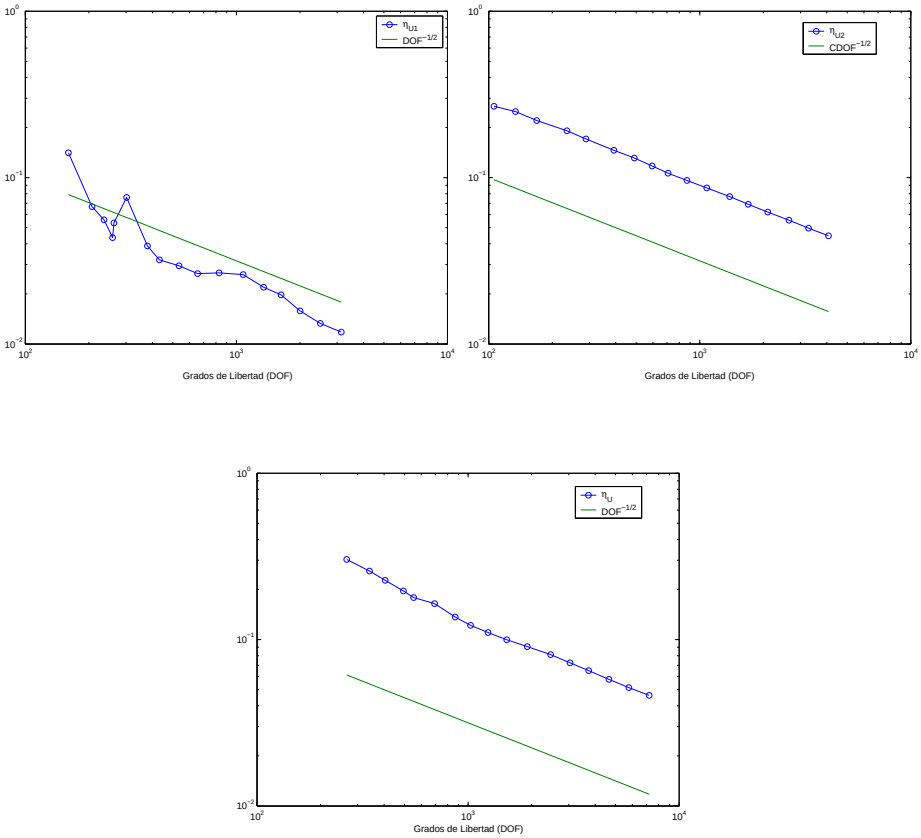


Figura 5.3: Error en Ω_1 (izquierda), en Ω_2 (derecha) y Error global (abajo)

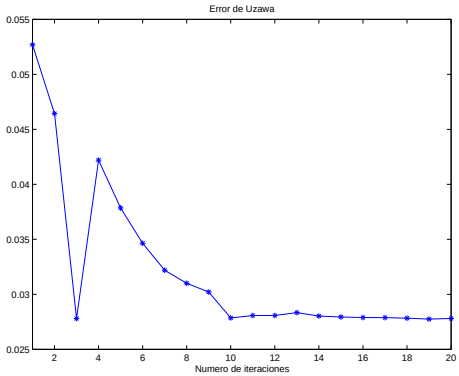


Figura 5.4: Error de Uzawa

iter	DOF1	DOF2	$\|P_j - P_{j-1}\|_{\mathbb{M}_{12}}$
1	53	83	0.350776
2	31	70	0.988743
3	70	74	1.85776
4	160	106	0.303222
5	207	134	0.258318
6	236	169	0.227254
7	259	235	0.196116
8	263	289	0.178851
9	302	392	0.164578
10	379	490	0.136774
11	432	595	0.121828
12	535	708	0.110289
13	655	870	0.0997682
14	828	1080	0.0907472
15	1077	1386	0.0812183
16	1346	1696	0.0725177
17	1628	2102	0.0650993
18	2004	2647	0.0576839
19	2500	3279	0.051445
20	3140	4074	0.0462323

Tabla 5.1: Tabla de resultados para la descomposición en dos subdominios. Notación: `iter`, iteración del AMUADD; `DOFi`, número de grados de libertad en Ω_i. $\|P_j - P_{j-1}\|_{\mathbb{M}_{12}}$, error de Uzawa.

A continuación presentamos el listado del código del ejemplo presentado que como ya referimos está implementado en Freefem++.

```
int inside = 2; int outside = 1;
border a(t=1,2){x=t;y=0;label=outside;};
border b(t=0,1){x=2;y=t;label=outside;};
border c(t=2,0){x=t;y=1;label=outside;};
border d(t=1,0){x = 1-t; y = t;label=inside;};
border e(t=0, 1){ x= 1-t; y = t;label=inside;};
border e1(t=pi/2,2*pi){ x= cos(t); y = sin(t);label=outside;};
int n=1,i=0;

mesh th =buildmesh( a(5*n) + b(5*n) + c(10*n) + d(5*n));
mesh TH = buildmesh( e(5*n) + e1(25*n) );
plot(th,TH,wait=1,ps="schwarz-no-th.eps");

fespace vh(th,P1);
fespace VH(TH,P1);
fespace Nh(th,P0);
fespace NH(TH,P0);
Nh rho;
NH RHO;
vh u=0,v,ua=0;
VH U,V,UA=0;
vh lambda=0;
real error=0.1; real EstErro1=0; real EstErro2=0; real EstErro=0;
```

```
problem PB(U,V,init=i,solver=LU) =
        int2d(TH)(dx(U)*dx(V)+dy(U)*dy(V))
        + int2d(TH)( 100*x*dx(U)*V-100*y*dy(U)*V)
        + int1d(TH,inside)(lambda*V)+ on(outside,U= 1);


problem pb(u,v,init=i,solver=Cholesky) =
        int2d(th)(dx(u)*dx(v)+dy(u)*dy(v))
        + int2d(th)( -v) + int1d(th,inside)(-lambda*v)
        + on(outside,u = 1 );


varf indicator2(uu,chiK1) =
        intalledges(th)(chiK1*lenEdge*square(jump(N.x*dx(u)
        + N.y*dy(u)))/2)
        +int2d(th)(chiK1*square(hTriangle*(1+dxx(u)+dyy(u))));


varf indicator1(UU,chiK2) =
        intalledges(TH)(chiK2*lenEdge*square(jump(N.x*dx(U)
        + N.y*dy(U)))/2)
        +int2d(TH)(chiK2*square(hTriangle*(dxx(U)+dyy(U)
                              -100*x*dx(U)+ 100*y*dy(U))));
```

```
for ( i=0 ;i<20; i++)
{
   PB;
   pb;
   lambda = lambda - (u-U)/2;
   RHO[] = indicator1(0,NH);
   EstErro1=sqrt(RHO[].sum);
   cout<< "EstErro1="<<EstErro1<< endl;
   rho[] = indicator2(0,Nh);
   EstErro2=sqrt(rho[].sum);
   cout<< "EstErro2="<<EstErro2<< endl;
   EstErro=sqrt(rho[].sum + RHO[].sum);
   cout<< "EstErro="<<EstErro<< endl;
   plot(th,wait=1);
   plot(TH,wait=1);
   TH=adaptmesh(TH,U, err=error);
   th=adaptmesh(th,u, err=error);
   plot(th,wait=1);
   plot(TH,wait=1);
   U=U;
   u=u;
   RHO=RHO;
   rho=rho;
```

```
    error=error*0.8;
  }
   plot(TH,th,ps="schwarz-no-thTH.eps",wait=1);
   plot(U,u,ps="schwarz-no-uU.eps",value=1);
   plot(U,ps="schwarz-U-circf.eps",value=1);
   plot(u,ps="schwarz-u-rectf.eps",value=1);
```

Capítulo 6

Algoritmo Paralelo

6.1. Introducción

Actualmente hay necesidad de sistemas de computación más rápidos y eficientes debido a la demanda de la computación de alto nivel en distintas áreas como en mecánica, biomecánica, meteorología e ingeniería, entre otras. Esto llevó a la aparición de máquinas con nuevas arquiteturas y con ello la necesidad de adaptar algoritmos clásicos a esta nueva realidad, así como desarrollar nuevos algoritmos más eficientes que saquen partido de estas arquiteturas. Un ejemplo de esto son los computadores con multi-procesadores o procesadores multi-núcleo que nos permiten hacer procesamiento paralelo. La computación paralela se basa en que los problemas se pueden dividir en partes más pequeñas que pueden resolverse de forma simultánea. Dentro del procesamiento paralelo podemos distinguir dos modelos básicos de organización de memoria [19], [35]: memoria distribuida y memoria compartida. En los sistemas de memoria distribuida, cada procesador posee su propia memoria local y es necesario el uso de mensajes para la comunicación entre los procesadores. A la hora de medir la eficiencia en estos sistemas los

tiempos de comunicación son un factor importante a tener en cuenta, así como la sincronización entre los distintos procesadores. En cambio, los sistemas de memoria compartida se caracterizan por la existencia de una memoria global a la que acceden todos los procesadores. En este modelo, la sincronización se hace por el control de las operaciones de escritura y lectura hecha por los procesos. Tiene la ventaja de la rapidez de acceso a los datos y como desventaja la limitación entre los caminos entre los procesadores y la memoria, lo que disminuye la escalabilidad (ver Definición 6.2) del sistema. El objetivo de este capítulo es mostrar que nuestro algoritmo AMUADD tiene una estrategia de paralelización simple, no existiendo necesidad de comunicación ya que el cálculo de la solución en cada subdominio se hace de forma independiente. Así, de acuerdo con la descomposición del dominio elegida, cada procesador se va hacer cargo del cálculo de una secuencia de instrucciones correspondientes al cálculo de la solución del problema en cada subdominio. En nuestro caso, para la implementación paralela, utilizamos una plataforma Unix en lenguaje C++ [20], [40], en un ordenador (modelo de memoria compartida) con 4 núcleos. Para paralelizar el algoritmo se ha usado el OpenMP ([41], [45], [51]), que es un modelo portátil, escalable, que proporciona una interfaz simple y flexible a los programadores que desarrollan algoritmos paralelos para sistemas de memoria compartida y son usados en plataformas que van del ordenador portátil al super ordenador y, del Unix al Windows NT. El OpenMP ([10], [11]) es un conjunto de compiladores pragmas", directivas, llamadas de funciones y variables de entorno que explícitamente instruyen al compilador como y donde insertar hilos en la aplicación.

6.2. Terminología Paralela

A continuación presentamos algunas definiciones clásicas ([19], [27], [35]) en el cómputo de la computación paralela para que se pueda comprender mejor y hacer el análisis de la eficiencia del algoritmo desarrollado, AMUADD paralelo.

Definición 6.1. La Eficiencia de un algoritmo paralelo, ejecutado en una topología con p procesadores es el valor dado por:

$$\texttt{Eficiencia} = \frac{T(1)}{p * T(p)} \tag{6.1}$$

donde $T(p)$ representa el tiempo de ejecución del algoritmo en p.

Junto al concepto de eficiencia de un algoritmo paralelo, está también el concepto de speed-up de un algoritmo, el cual se define por:

$$\texttt{Speed-up} = \frac{T_{seq}}{T_{par}} \tag{6.2}$$

donde T_{par} representa el tiempo de ejecución del algoritmo en paralelo y T_{seq} representa el tiempo necesario para que un solo procesador ejecute el mejor algoritmo secuencial.

Obsérvese que de las relaciones (6.1) y (6.2) se puede escribir,

$$\texttt{Eficiencia} = \frac{\texttt{Speed-up}}{p} \tag{6.3}$$

Además, en teoría

$$T_{par} \geq \frac{T_{seq}}{p}$$

por lo que

$$\texttt{Speed-up} \leq p \ \texttt{y Eficiencia} \leq 1$$

En el caso ideal (`Speed-up` $= p$) la eficiencia sería máxima y tendría el valor 1. En algunas situaciones existen problemas en los que el mejor algoritmo secuencial, no se puede paralelizar directamente, implicando dos algoritmos distintos en el cálculo del speed-up y en la eficiencia. Este problema no surge en nuestro algoritmo ya que el AMUADD se ha paralelizado de una forma sencilla dando origen al algoritmo expuesto en este capítulo.

Otros conceptos que son importantes en el procesamiento paralelo son la escalabilidad (Scalability) del algoritmo y el balance de carga (Load Balancing) que se definen a continuación.

Definición 6.2. Un sistema computacional paralelo se dice escalable si la aceleración conseguida aumenta proporcionalmente al número de procesadores.

Definición 6.3. El balance de carga consiste en la distribución equilibrada de tareas entre los procesadores, para garantizar una ejecución eficiente del programa paralelo.

Otro concepto importante cuando se habla de computación paralela es el balance de un algoritmo. Se dice que un algoritmo paralelo está balanceado, cuando en cada paso del algoritmo, cada uno de los procesadores utilizados tiene el mismo volumen de computación y comunicación a realizar.

6.3. Resultados

Los resultados presentados en el capítulo §3 se han obtenido usando la llamada computación secuencial, en que una única instrucción se ejecuta en un

momento dado de tiempo en un único CPU (Central Processing Unit). A continuación presentamos los resultados obtenidos con la resolución del Problema 3.1 pero ahora con el algoritmo paralelo del AMUADD. La paralelización del algoritmo consiste en introducir el código secuencial correspondiente al cálculo de la solución del problema en cada uno de los subdominios dentro de la instrucción $\sharp pragma \quad omp \quad section$ del OpenMP, que informa al compilador que esa sección de código va a ser atribuida a un hilo, y que va a ser ejecutada simultáneamente con los otros hilos. Respecto a las variables globales no hay ningún cambio, ya que estamos utilizando un sistema de memoria compartido y por lo tanto todas las variables acceden a la memoria global del ordenador.

6.3.1. Descomposición en 2 subdominios

Se considera ahora la resolución del Problema 3.1 con el algoritmo paralelo usando la descomposición simétrica (ver Ejemplo 3.1) del dominio Ω en dos subdominios Ω_1 y Ω_2, con iguales áreas. Obsérvese en la Figura 6.1 el funcionamiento del sistema cuando se considera un único procesador (arriba) y cuando se consideran 2 procesadores (abajo). El uso del algoritmo paralelo con 1 procesador corresponde a resolver el problema de forma secuencial mientras que cuando se consideran 2, el cálculo se hace en paralelo. En este caso, cada procesador tiene el mismo problema secuencial para solucionar y lo hace de forma independiente. Se observa que cuando se considera 1 procesador, en este caso, todos los cálculos se hacen en el CPU 4 mientras que con 2 procesadores los cálculos se hacen en paralelo y se han atribuido al CPU 1 y al CPU 4.

Se considera ahora la resolución del mismo problema con la descomposición no simétrica (Ejemplo 3.2) del dominio Ω en dos subdominios Ω_1 y Ω_2.

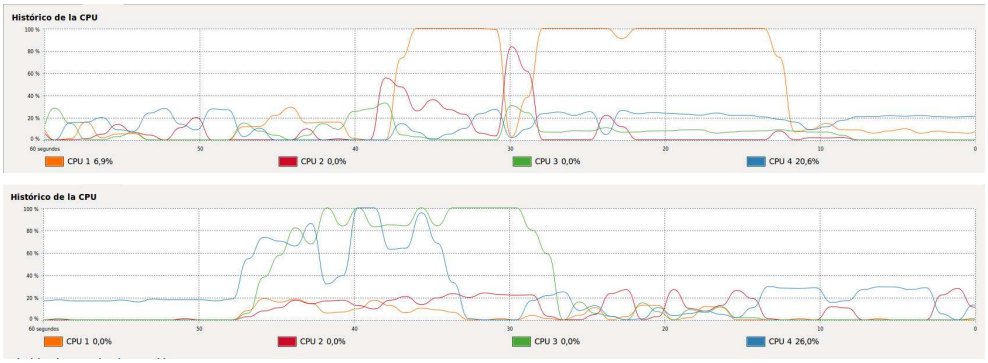


Figura 6.1: Funcionamiento del sistema usando un procesador (arriba) y dos procesadores (abajo).

Con esta descomposición el subdominio Ω_1 tiene menor área que Ω_2. Se observa en la Figura 6.2 el funcionamiento del sistema cuando se considera un único procesador (arriba) y cuando se consideran dos procesadores (abajo). Tal como en el ejemplo anterior, cuando se consideran 2 procesadores, cada uno de ellos resuelve el mismo problema secuencial para obtener la solución en cada subdominio. Se observa que con un procesador el CPU donde se realiza todo el cálculo es el 1, mientras que cuando se usan 2 procesadores el cálculo del mismo problema secuencial se hace en paralelo en el CPU 2 y en el CPU 3.

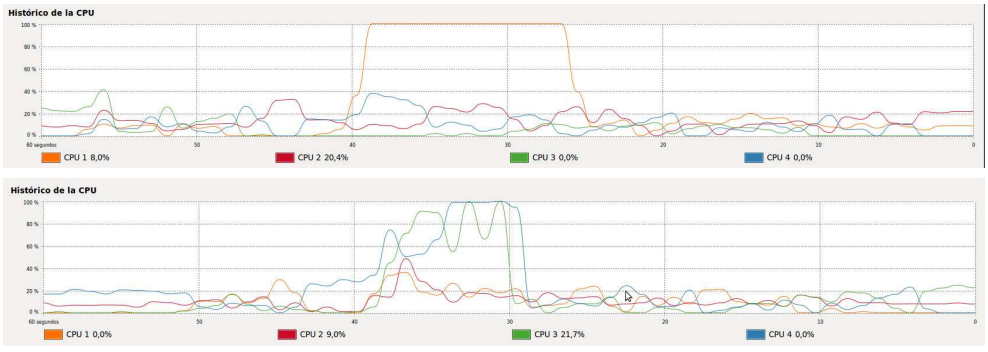


Figura 6.2: Funcionamiento del sistema usando un procesador (arriba) y dos procesadores (abajo).

En la Tabla 6.1 presentamos los tiempos gastados considerando las dos descomposiciones presentadas, la descomposición simétrica (2DS) y la no simétrica (2DNS) en dos subdominios con 1 y con 2 procesadores. Presentamos también en la Tabla 6.2 los correspondientes valores del speed-up y de la eficiencia obtenida con cada descomposición.

$Descomp$	$T(1)$	$T(2)$
$2DS$	24.63	18.36
$2DNS$	13.61	8.57

Tabla 6.1: Tabla de los tiempos para la descomposiciones en dos subdominios. Notación: *descomp*, descomposición utilizada; $T(p)$, tiempo gasto en segundos con p procesadores.

$Descomp$	$Speed-up$	$Eficiencia$
$2DS$	1.34	0.67
$2DNS$	1.59	0.80

Tabla 6.2: Tabla con los valores de Speed-up y Eficiencia.

Respecto a la descomposición del dominio Ω en 2 subdominios, se observa que la descomposición no simétrica $2DNS$ es la que presenta un mayor valor de speed-up y un mayor valor de eficiencia, frente a la descomposición simétrica $2DS$. En conclusión, a la hora de elegir una descomposición en 2 subdominios para paralelizar deberíamos escoger la no simétrica porque es la más eficiente, presenta 80 % de eficiencia, y es también aquella con que se tarda menos en resolver nuestro problema test.

6.3.2. Descomposición en 4 subdominios

Se considera ahora la resolución del Problema 3.1 con el algoritmo paralelo usando la descomposición no simétrica (ver Ejemplo 3.5) del dominio Ω en cuatro subdominios Ω_1, Ω_2, Ω_3 y Ω_4 con diferentes áreas. Obsérvese en la Figura 6.3 el funcionamiento del sistema cuando se consideran 1, 2, 3 y 4 procesadores. El uso del algoritmo paralelo con 1 procesador corresponde a resolver el problema de forma secuencial y como tenemos 4 subdominios vamos aumentando el número procesadores hasta 4, para que al final, cada uno de ellos se quede con el cálculo de una solución U_i, para $i = 1, 2, 3, 4$. Tal como en los ejemplos anteriores, cada procesador tiene el mismo problema secuencial para solucionar y lo hace de forma independiente.

En la Tabla 6.3 presentamos los tiempos empleados en la resolución del problema con el AMUADD paralelo con 1, 2, 3 y 4 procesadores y, en la Tabla 6.4, los correspondientes valores de speed-up y eficiencia obtenidos.

$T(1)$	$T(2)$	$T(3)$	$T(4)$
23.78	14.96	13.50	12.53

Tabla 6.3: Tabla de los tiempos para la descomposición en 4 subdominios no simétricos. Notación: $T(p)$, tiempo usado en segundos con p procesadores.

$T(p)$	$Speed-up$	$Eficiencia$
$T(2)$	1.59	0.80
$T(3)$	1.76	0.59
$T(4)$	1.90	0.48

Tabla 6.4: Tabla del Speed-up y Eficencia en el caso de la descomposición no simétrica en 4 sudominios.

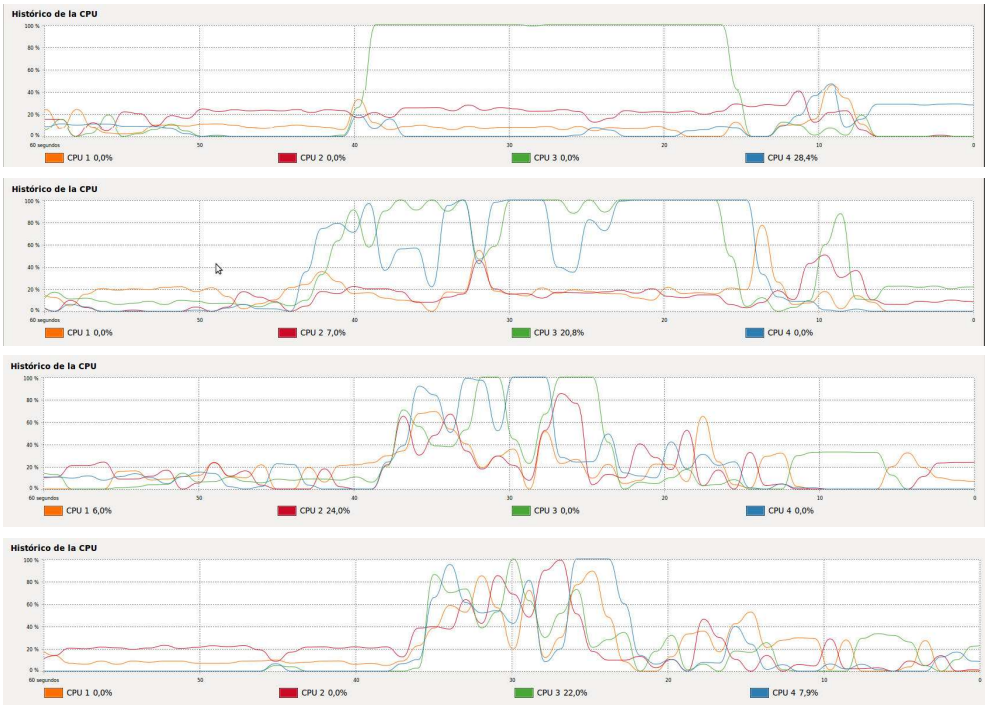


Figura 6.3: Funcionamiento del sistema usando un procesador (arriba), dos procesadores (segunda), tres procesadores (tercera) y cuatro procesadores (abajo).

Se considera ahora la resolución del Problema 3.1 con el algoritmo paralelo usando la descomposición simétrica (ver Ejemplo 3.4) del dominio Ω en cuatro subdominios Ω_1, Ω_2, Ω_3 y Ω_4 con iguales áreas. Obsérvese, en la Figura 6.4, el funcionamiento del sistema cuando se consideran 1 y 4 procesadores. El uso del algoritmo paralelo con 1 procesador corresponde a resolver el problema de forma secuencial y como no se ha observado una gran diferencia entre los tiempos de cálculo para 2 y 3 procesadores presentamos los resultados obtenidos con 1 y con 4 procesadores. Tal como en la descomposición anterior, el interés en usar 4 procesadores está en que con esta tipología cada uno de ellos se queda con el cálculo de una solución U_i, para $i = 1, 2, 3, 4$. Como en los ejemplos anteriores,

cada procesador tiene el mismo problema secuencial para solucionar y lo hace de forma independiente.

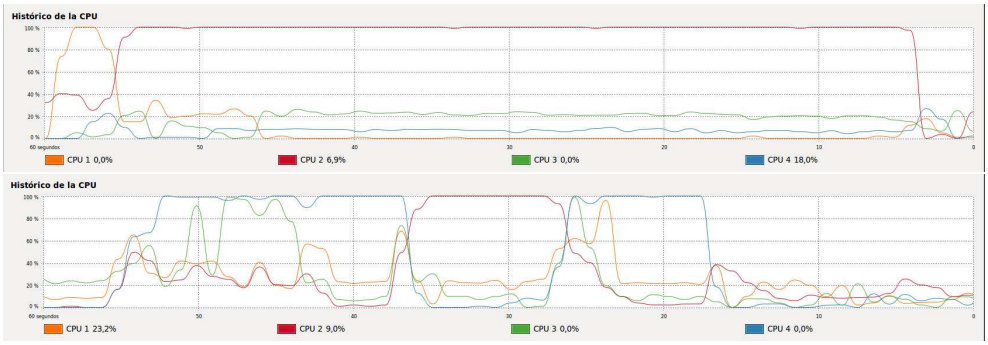


Figura 6.4: Funcionamiento del sistema usando un procesador (arriba) y cuatro procesadores (abajo).

En la Tabla 6.5 presentamos los tiempos empleados en la resolución del problema con el AMUADD paralelo con 1, 2, 3 y 4 procesadores y, en la Tabla 6.6, los correspondientes valores de speed-up y eficiencia obtenidos.

$T(1)$	$T(2)$	$T(3)$	$T(4)$
54.31	39.93	38.05	37.55

Tabla 6.5: Tabla de los tiempos para la descomposición en 4 subdominios simétricos. Notación: $T(p)$, tiempo empleado en segundos con p procesadores.

$T(p)$	$Speed-up$	$Eficiencia$
$T(2)$	1.36	0.68
$T(3)$	1.43	0.48
$T(4)$	1.45	0.36

Tabla 6.6: Tabla del Speed-up y Eficencia en el caso de la descomposición simétrica en 4 sudominios.

Analizando los valores speed-up y eficiencia para nuestro problema se puede concluir con base en los resultados obtenidos con el AMUADD paralelo, que el algoritmo paralelo es bastante eficiente cuando se usan apenas dos procesadores y en descomposiciones no simétricas. Si atendemos a las características del problema es fácil explicar las razones para que esto suceda. Con la descomposición en cuatro subdominios y usando 4 procesadores el algoritmo no está balanceado porque vamos a tener por lo menos un subdominio donde se hacen muchas adaptaciones de mallado en relación a los demás subdominios. En consecuencia, hay un procesador que va a tener más cálculo que los demás. La diferencia de eficiencia para $p = 4$ que se observa entre la descomposición simétrica y la no simétrica se explica por el hecho de que con la no simétrica se consigue un algoritmo más balanceado ya que se atenúan las diferencias entre los volúmenes de computación realizadas por cada procesador. Respecto a la rapidez de convergencia se observa que en ambas descomposiciones los valores de speed-up aumentan con el número de procesadores.

A continuación se presenta el listado del programa principal del AMUADD paralelo.

6.4. Código del Algoritmo

```
#include<omp.h>
#include<string.h>
#include<math.h>
#include<unistd.h> // Para medir el tiempo de CPU
#include<sys/times.h> // Para medir el tiempo de CPU
```

```
#include"mesh.h"
#include"mesh1d.h"
#include "matrix.h"
#include"problem1d.h"
#include "finite_element.h"
#include"difusion_2d.h"
#include "difusion_2d_boundary.h"
#include"problem.h"
#include "vect_d.h"
#include "fe_field.h"
#include "fe_field1d.h"
#include "field.h"
#include "const_field.h"
#include"matriztridiag.h"

using namespace std;

typedef problem <difusion_2d, difusion_2d_boundary, matrix, field,
fe_field1d> PROBLEM;

int NR; double dt=1.; double tm; double OutValue=0.;

double null(double *x){return 0;}
field null_value=null;
```

```
// fuente de calor
 double f(double *x) {
 double alpha=100.;
 double beta=0.36388;
 double r=sqrt(x[0]*x[0]+x[1]*x[1]);
 return 2.*(1.+ alpha*(r-beta)*atan(alpha*(r-beta)
            + atan(alpha*r)));
 }
 field fuente=f;

// valores del coeficiente de difusion
 double df(double *x) {
 double alpha=100.;
 double beta=0.36388;
 double r=sqrt(x[0]*x[0]+x[1]*x[1]);
 return 1./alpha + alpha*(r-beta)*(r-beta);
 }
 field difusion1=df; field difusion2=df;
 field difusion3=df; field difusion4=df;

 // valor del flujo en la frontera
 const_field flow=1.;

 // valor en la frontera con condicion de Dirichlet
 const_field dirichlet=0.;
```

```
int main() {
 //
 // declara las mallas la lee del fichero mesh.msh
 // (formato freefem) y la debuja con MEDIT
 double Lx=0.3,Ly=0.3;  // dimensiones del dominio
 double area_dom1 = Lx * Ly;   // Area del dominio 1
 int nrows=4,ncols=4; // numero de filas y columnas
 //Paso espacio en direcciones x e y
 double hx=Lx/(ncols-1);double hy=Ly/(nrows-1);
 double xorig[2]={0,0};
 mesh  TH1(nrows,ncols,hx,hy,xorig,1,-2,-3,4);
 #ifdef DIBUJAR
  TH1.plot();
 #endif
 Lx=0.7;
 double area_dom2 = Lx * Ly;  // Area del dominio 2
 ncols=8;
 hx=Lx/(ncols-1);
 xorig[0]=0.3;xorig[1]=0.;
 mesh TH2(nrows,ncols,hx,hy,xorig,1,2,-3,-4);
 #ifdef DIBUJAR
  TH2.plot();
 #endif
 nrows=11;
```

```
hy=Ly/(nrows-1);
mesh1d TH12(nrows,hy,xorig);

Lx=0.3; Ly=0.7; // dimensiones del dominio
double area_dom3 = Lx * Ly;  // Area del dominio 3
nrows=8;ncols=4; // numero de filas y columnas
//Paso espacio en direcciones x e y
hx=Lx/(ncols-1);hy=Ly/(nrows-1);
xorig[0]=0.;xorig[1]=0.3;
mesh TH3(nrows,ncols,hx,hy,xorig,-1,-2,3,4);
#ifdef DIBUJAR
 TH3.plot();
#endif
ncols=11;
hx=Lx/(ncols-1);
mesh1d TH13(ncols,hx,xorig);

Lx=0.7; Ly=0.7; // dimensiones del dominio
double area_dom4 = Lx * Ly;   // Area del dominio 4
nrows=8;ncols=8; // numero de filas y columnas
//Paso espacio en direcciones x e y
hx=Lx/(ncols-1);hy=Ly/(nrows-1);
xorig[0]=0.3;xorig[1]=0.3;
mesh TH4(nrows,ncols,hx,hy,xorig,-1,2,3,-4);
#ifdef DIBUJAR
```

```
 TH4.plot();
#endif
nrows=11;
hy=Ly/(nrows-1);
mesh1d TH34(nrows,hy,xorig);
ncols=11;
hx=Lx/(ncols-1);
mesh1d TH24(ncols,hx,xorig);

// Area del dominio; se usa en el control del error
double AreaDelDominio = area_dom1 + area_dom2
                        + area_dom3 + area_dom4 ;

int nmeshmax=30;
mesh th1[nmeshmax]; th1[0]=TH1;
mesh th2[nmeshmax]; th2[0]=TH2;
mesh th3[nmeshmax]; th3[0]=TH3;
mesh th4[nmeshmax]; th4[0]=TH4;

//
// Codigo que pertenece al problema que esta a
// ser resolvido en cada subdominio

// Dominio 1
// s1d: simplex 1d (intervalo), 2 nodos(P1), 2"caras"
```

```
finite_element fe1d_1(s1d,2,1,2,2);

// s2d: simplex 2d (triangulo), 3 nodos(P1), 3 caras
finite_element fe2d_1(s2d,3,3);
// elemento frontera
difusion_2d_boundary BoundaryEquations_1(&fe1d_1);

// elemento para problemas elipticos segundo orden
difusion_2d FlowEquations_1(&fe2d_1);
//
// Dominio 2
// s1d: simplex 1d (intervalo), 2 nodos(P1), 2"caras"
finite_element fe1d_2(s1d,2,1,2,2);
// s2d: simplex 2d (triangulo), 3 nodos(P1), 3 caras
finite_element fe2d_2(s2d,3,3);
// elemento frontera
difusion_2d_boundary BoundaryEquations_2(&fe1d_2);
// elemento para problemas elipticos segundo orden
difusion_2d FlowEquations_2(&fe2d_2);
//
// Dominio 3
// s1d: simplex 1d (intervalo), 2 nodos(P1), 2"caras"
finite_element fe1d_3(s1d,2,1,2,2);
// s2d: simplex 2d (triangulo), 3 nodos(P1), 3 caras
finite_element fe2d_3(s2d,3,3);
```

```
// elemento frontera
difusion_2d_boundary BoundaryEquations_3(&fe1d_3);
// elemento para problemas elipticos segundo orden
difusion_2d FlowEquations_3(&fe2d_3);
//
// Dominio 4
// s1d: simplex 1d (intervalo), 2 nodos(P1), 2"caras"
finite_element fe1d_4(s1d,2,1,2,2);
// s2d: simplex 2d (triangulo), 3 nodos(P1), 3 caras
finite_element fe2d_4(s2d,3,3);
// elemento frontera
difusion_2d_boundary BoundaryEquations_4(&fe1d_4);
// elemento para problemas elipticos segundo orden
difusion_2d FlowEquations_4(&fe2d_4);

// crea las matrices de los sistemas
matriztridiag *A12, *A13,*A34, *A24 ;
A12=NULL;
A13=NULL;
A34=NULL;
A24=NULL;
double rho;
cout << " parametro de Uzawa rho = ? "<< endl;
cin>> rho;
double tol=10;
```

```
int jmax=20;
double GlobalError12=1.;
double GlobalError13=1.;
double GlobalError34=1.;
double GlobalError24=1.;
double GlobalError1=1.;
double GlobalError2=1.;
double GlobalError3=1.;
double GlobalError4=1.;

double ua=0;
double ub=0;

int neq12=TH12.get_numnp();
int neq13=TH13.get_numnp();
int neq24=TH24.get_numnp();
int neq34=TH34.get_numnp();

fe_field1d lambda1(&TH12);
fe_field1d lambda2(&TH12);
fe_field1d lambda5(&TH13);
fe_field1d lambda6(&TH13);
fe_field1d lambda7(&TH24);
fe_field1d lambda8(&TH24);
fe_field1d lambda3(&TH34);
```

```
fe_field1d lambda4(&TH34);

vect_d lam1(neq12);  // aqui almacenaremos los valores impuestos
vect_d lam2(neq13);  // aqui almacenaremos los valores impuestos
vect_d lam3(neq24);  // aqui almacenaremos los valores impuestos
vect_d lam4(neq34);  // aqui almacenaremos los valores impuestos

// solucion inicial de problema1d
for(int i=0;i<neq12;i++) lam1[i]=0;
lambda1=lam1;
lambda2=lam1;

for(int i=0;i<neq13;i++) lam2[i]=0;
lambda5=lam2;
lambda6=lam2;

for(int i=0;i<neq24;i++) lam3[i]=0;
lambda7=lam3;
lambda8=lam3;

for(int i=0;i<neq34;i++) lam4[i]=0;
lambda3=lam4;
lambda4=lam4;

int lmesh1,lmesh2,lmesh3,lmesh4;
```

```
int linit1=1,linit2=1,linit3=1,linit4=1;

fe_field1d null_field1d1(&TH12);
fe_field1d null_field1d2(&TH24);
fe_field1d null_field1d3(&TH13);
fe_field1d null_field1d4(&TH34);

vect_d empty1(neq12);
vect_d empty3(neq13);
vect_d empty2(neq24);
vect_d empty4(neq34);

null_field1d1=empty1;
null_field1d2=empty2;
null_field1d3=empty3;
null_field1d4=empty4;

fe_field fe_U1(&TH1);
fe_field fe_U2(&TH2);
fe_field fe_U3(&TH3);
fe_field fe_U4(&TH4);

// Para obtener el tiempo de ejecucion del programa total
clock_t t1, t2;
```

```
// Estructura para medir el tiempo de CPU (man 2 times)
struct tms crono;

// Anoto el instante t1
t1 = times(&crono);

int num_threads; // Numero de hilos que se lanzan

for(int j=0;j<jmax;j++)
{
 tol*=0.9;

 #pragma omp parallel sections
 {
  num_threads = omp_get_num_threads();
  //
  // PROBLEMA 1
  //
  #pragma omp section
  {
   cout << "Problema 1 en el hilo " << omp_get_thread_num()
        << " de " <<  num_threads << endl;
   for (lmesh1=linit1;lmesh1<nmeshmax;lmesh1++)
   {
    int numel1=th1[lmesh1-1].get_numel();
```

```
vect_d ErrInd1(numel1);

// declara un tipo de problema problema
PROBLEM Flow1(&th1[lmesh1-1]);

// crea la matriz del sistema
matrix *A1 = new matrix;

// Calcula la estructura de la matriz
// y ensambla los terminos correspondientes del
// material con numero de referencia=0,
// este es el valor de referencia por defecto.
Flow1.set_problem_matrix(&FlowEquations_1, A1, difusion1);

int neq1=A1->get_neq();
cout << " Num de ecuaciones del sistema1 = "<< neq1 <<endl;
vect_d b1(neq1); //aqui almacenamos el segundo miembro
vect_d u10(neq1);//aqui almacenaremos los valores impuestos
vect_d u1(neq1); //aqui almacenamos la solucion

// Calcula el valor de la solucion en
//las fronteras de dirichlet fijas
Flow1.set_dirichlet_boundary(u10, dirichlet, 1);
Flow1.set_dirichlet_boundary(u10, dirichlet, 4);
for (int i=0;i<b1.size();i++) b1[i]=0.;
```

```
 Flow1.set_problem_second_member(&FlowEquations_1,b1,fuente,
                                  difusion1,null_value,u10);

 // Anade contribucion frontera
 Flow1.add_boundary_second_member(&BoundaryEquations_1,
                                  b1, lambda1, -2);
 Flow1.add_boundary_second_member(&BoundaryEquations_1,
                                  b1, lambda5, -3);

// Modifica el sistema:matriz y segundo miembro
Flow1.set_dirichlet_boundary(A1,b1,u10,1);
Flow1.set_dirichlet_boundary(A1,b1,u10,4);

// Resuelve el sistema por el metodo de Gradiente conjugado
//con precondicionador diagonal
int it = Flow1.solve(b1,u1);

// Calcula el error
GlobalError1=Flow1.get_error(&FlowEquations_1,
                             &BoundaryEquations_1,
                             ErrInd1,difusion1,
                             fuente,null_field1d1,
                             lambda1,u1);
cout <<" Num de refinamientos realizados malla1= "<<lmesh1-1
                                                   <<endl;
```

```
    cout <<" ERROR GLOBAL ESTIMADO no dominio 1= "<<GlobalError1
                                                    << endl;


    TH1=th1[lmesh1-1];
    fe_U1=u1;
    if (GlobalError1 <(tol*area_dom1/AreaDelDominio)) break;
    else    //refina la malla
    th1[lmesh1].refine(th1[lmesh1-1],ErrInd1);
    delete A1;
  } // for (lmesh1=...)
#ifdef DIBUJAR
   fe_U1.plot();
#endif
  linit1=lmesh1;
 } //pragma omp section


  // PROBLEMA 2
#pragma omp section
{
 cout << "Problema 2 en el hilo " << omp_get_thread_num()
      << " de " <<  num_threads << endl;


 for(lmesh2=linit2;lmesh2<nmeshmax;lmesh2++)
 {
  int numel2=th2[lmesh2-1].get_numel();
```

```
vect_d ErrInd2(numel2);

// Declara un tipo de problema problema
PROBLEM Flow2(&th2[lmesh2-1]);

matrix *A2 = new matrix; // crea la matriz del sistema

// Calcula la estructura de la matriz y ensambla
//los terminos correspondientes del material con numero
// de referencia=0, este es el valor de referencia por defecto
Flow2.set_problem_matrix(&FlowEquations_2, A2, difusion2);
int neq2=A2->get_neq();

cout << " Numero de ecuaciones del sistema 2 = "<< neq2 <<endl;
vect_d b2(neq2); // aqui almacenamos el segundo miembro
vect_d u20(neq2);// aqui almacenaremos los valores impuestos
vect_d u2(neq2); // aqui almacenamos la solucion

// Calcula el valor de la solucion en las fronteras
//de dirichlet fijas
Flow2.set_dirichlet_boundary(u20, dirichlet, 1);
Flow2.set_dirichlet_boundary(u20, dirichlet, 2);
for (int i=0;i<b2.size();i++) b2[i]=0.;
Flow2.set_problem_second_member(&FlowEquations_2, b2,fuente,
                                difusion2,null_value,u20);
```

```
//Anade contribucion frontera
Flow2.add_boundary_second_member(&BoundaryEquations_2,
                                 b2,lambda7,-3);
Flow2.add_boundary_second_member(&BoundaryEquations_2,
                                 b2,lambda2,-4);

// Modifica el sistema:matriz y segundo miembro
Flow2.set_dirichlet_boundary(A2,b2,u20,1);
Flow2.set_dirichlet_boundary(A2,b2,u20,2);
// Resuelve el sistema por el metodo de Gradiente conjugado
//con precondicionador diagonal
int it = Flow2.solve(b2,u2);

// Calcula el error
GlobalError2= Flow2.get_error(&FlowEquations_2,
                              &BoundaryEquations_2,
                              ErrInd2, difusion2,
                              fuente,null_field1d1,
                              lambda2,u2);
cout << " Num de refinamientos realizados malla 2 = "<<lmesh2-1
                                                      <<endl;
cout << " ERROR GLOBAL ESTIMADO no dominio 2= "<< GlobalError2
                                                      <<endl;
TH2=th2[lmesh2-1];
fe_U2=u2;
```

```
    if (GlobalError2 < (tol*area_dom2/AreaDelDominio)) break;
    else    //refina la malla
      th2[lmesh2].refine(th2[lmesh2-1],ErrInd2);
    delete A2;
    } // for (lmesh2= ...)
    #ifdef DIBUJAR
     fe_U2.plot();
    #endif
    linit2=lmesh2;
   } //pragma omp section

    // PROBLEMA 3
    //
  #pragma omp section
  {
   cout << "Problema 3 en el hilo "<< omp_get_thread_num()
        << " de " <<  num_threads << endl;

   for(lmesh3=linit3;lmesh3<nmeshmax;lmesh3++)
   {
    int numel3=th3[lmesh3-1].get_numel();
    vect_d ErrInd3(numel3);

    // Declara un tipo de problema problema
    PROBLEM Flow3(&th3[lmesh3-1]);
```

```
// Crea la matriz del  sistema
matrix *A3 = new matrix ;

// Calcula la estructura de la matriz y ensambla
//los terminos correspondientes del material con numero
// de referencia=0, este es el valor de referencia por defecto
Flow3.set_problem_matrix(&FlowEquations_3, A3, difusion3);
int neq3=A3->get_neq();
cout << " Num de ecuaciones del sistema 3 = "<< neq3 <<endl;
vect_d b3(neq3); // aqui almacenamos el segundo miembro
vect_d u30(neq3);// aqui almacenaremos los valores impuestos
vect_d u3(neq3);// aqui almacenamos la solucion

// Calcula el valor de la solucion en las fronteras
//de dirichlet fijas
Flow3.set_dirichlet_boundary(u30, dirichlet, 3);
Flow3.set_dirichlet_boundary(u30, dirichlet, 4);
for (int i=0;i<b3.size();i++) b3[i]=0.;
Flow3.set_problem_second_member(&FlowEquations_3, b3, fuente,
                                difusion3, null_value, u30);
// Anade contribucion frontera
Flow3.add_boundary_second_member(&BoundaryEquations_3,
                                 b3, lambda6,-1);
Flow3.add_boundary_second_member(&BoundaryEquations_3,
                                 b3, lambda3, -2);
```

```
// Modifica el sistema:matriz y segundo miembro
Flow3.set_dirichlet_boundary(A3,b3,u30,3);
Flow3.set_dirichlet_boundary(A3,b3,u30,4);
// Resuelve el sistema por el metodo de Gradiente conjugado
//con precondicionador diagonal
int it = Flow3.solve(b3,u3);

// Calcula el error
GlobalError3= Flow3.get_error(&FlowEquations_3,
                              &BoundaryEquations_3,
                              ErrInd3, difusion3,
                              fuente,null_field1d3,
                              lambda6,u3);

cout << " Num de refinamientos realizados malla 3 = "<<lmesh3-1
                                                      <<endl;

cout << " ERROR GLOBAL ESTIMADO no dominio 3= "<< GlobalError3
                                                   << endl;

TH3=th3[lmesh3-1];
fe_U3=u3;

if (GlobalError3 < (tol*area_dom3/AreaDelDominio)) break;
```

```
    else    //refina la malla
      th3[lmesh3].refine(th3[lmesh3-1],ErrInd3);
    delete A3;
   } // for (lmesh3= ...)
  #ifdef DIBUJAR
      fe_U3.plot();
  #endif
  linit3=lmesh3;
 } //pragma omp section

    // PROBLEMA 4

  #pragma omp section
  {
   cout << "Problema 4 en el hilo " << omp_get_thread_num()
   << " de " <<  num_threads << endl;

   for (lmesh4=linit4;lmesh4<nmeshmax;lmesh4++)
   {
    int numel4=th4[lmesh4-1].get_numel();
    vect_d ErrInd4(numel4);

    // Declara un tipo de problema problema
   PROBLEM Flow4(&th4[lmesh4-1]);
```

```
// Crea la matriz del sistema
matrix *A4 = new matrix;

// Calcula la estructura de la matriz y ensambla
//los terminos correspondientes del material con numero
//de referencia=0, este es el valor de referencia por defecto
Flow4.set_problem_matrix(&FlowEquations_4, A4, difusion4);

int neq4=A4->get_neq();

cout <<" Num de ecuaciones del sistema 4 = "<< neq4 <<endl;
vect_d b4(neq4); // aqui almacenamos el segundo miembro
vect_d u40(neq4);// aqui almacenaremos los valores impuestos
vect_d u4(neq4);// aqui almacenamos la solucion

// Calcula el valor de la solucion en las fronteras
//de dirichlet fijas
Flow4.set_dirichlet_boundary(u40, dirichlet, 2);
Flow4.set_dirichlet_boundary(u40, dirichlet, 3);
for (int i=0;i<b4.size();i++) b4[i]=0.;
Flow4.set_problem_second_member(&FlowEquations_4, b4, fuente,
                                difusion4, null_value, u40);

// Anade contribucion frontera
Flow4.add_boundary_second_member(&BoundaryEquations_4,
```

```
                                    b4, lambda8, -1);
Flow4.add_boundary_second_member(&BoundaryEquations_4,
                                    b4, lambda4, -4);

// Modifica el sistema:matriz y segundo miembro
Flow4.set_dirichlet_boundary(A4,b4,u40,2);
Flow4.set_dirichlet_boundary(A4,b4,u40,3);

// Resuelve el sistema por el metodo de Gradiente conjugado
//con precondicionador diagonal
int it = Flow4.solve(b4,u4);

// Calcula el error
GlobalError4= Flow4.get_error(&FlowEquations_4,
                              &BoundaryEquations_4,
                              ErrInd4, difusion4,
                              fuente,null_field1d4,
                              lambda8,u4);

cout << " Num de refinamientos realizados malla 4 = "<<lmesh4-1
                                                    <<endl;

cout << " ERROR GLOBAL ESTIMADO no dominio 4= "<< GlobalError4
                                                  << endl;
```

```
 TH4=th4[lmesh4-1];
 fe_U4=u4;
 if (GlobalError4 < (tol*area_dom4/AreaDelDominio)) break;
 else    //refina la malla
   th4[lmesh4].refine(th4[lmesh4-1],ErrInd4);
 delete A4;
 } // for (lmesh4= ...)
#ifdef DIBUJAR
    fe_U4.plot();
#endif
    linit4=lmesh4;

  } //pragma omp section
 } // pragma parallel sections


  //
  // Problema en la frontera interna 12
  //
  int numel12=TH12.get_numel();
  int neq12=TH12.get_numnp();
  vect_d ErrInd12(numel12);

  // Declara un tipo de problema en la frontera 12
  problem1d frontera(&TH12, &fe1d_1, &flow);
```

```
cout << " Num de ecuaciones del sistema 12 = "<< neq12 <<endl;
vect_d b(neq12);// aqui almacenamos el segundo miembro
vect_d u(neq12);// aqui almacenamos la solucion del problema
frontera.set_problem_second_member(fe_U1, fe_U2, b);

// calcula el error del algoritmo de Uzawa
double error=0.;
for(int i=0;i<b.size();i++) error+=b[i]*b[i];
cout << " error UZAWA 12 = "<< sqrt(error)<<endl;
for(int i=0;i<b.size();i++) b[i]=rho*b[i];
frontera.add_second_member(lambda1, b,0);
frontera.add_second_member(lambda1, b,1);

// Construye el sistema de ecuaciones con condiciones
//Dirichlet donde b es el segundo miembro del sistema
A12=frontera.set_problem_matriz(b,ua,ub);
A12->solve(b,u);

lambda1=u;
int n=u.size();
for(int i=1;i<n-1;i++) u[i]=(-1)*u[i];
lambda2=u;

GlobalError12=frontera.get_error1D( ErrInd12, u, fe_U1, fe_U2);
```

```
cout << "ERROR GLOBAL ESTIMADO en la frontera 12 = "
                                        << GlobalError12 << endl;

//
// Problema en la frontera interna 13
//

int numel13=TH13.get_numel();
int neq13=TH13.get_numnp();
vect_d ErrInd13(numel13);

// Declara un tipo de problema en la frontera 13
problem1d frontera13(&TH13, &fe1d_2, &flow);

cout << " Num de ecuaciones del sistema 13 = "<< neq13 <<endl;
vect_d b013(neq13);// aqui almacenamos el segundo miembro
vect_d u013(neq13);// aqui almacenamos la solucion del problema
frontera13.set_problem_second_member(fe_U1, fe_U3, b013);

// Calcula el error del algoritmo de Uzawa
double error13=0.;
for(int i=0;i<b013.size();i++) error13+=b013[i]*b013[i];
cout << " error UZAWA 13= "<< sqrt(error13)<<endl;
for(int i=0;i<b013.size();i++) b013[i]=rho*b013[i];
frontera13.add_second_member(lambda5, b013,0);
```

```
frontera13.add_second_member(lambda5, b013,1);

// Construye el sistema de ecuaciones con condiciones
//Dirichlet donde b es el segundo miembro del sistema
A13=frontera13.set_problem_matriz(b013,ua,ub);
A13->solve(b013,u013);

ambda5=u013;
int n13=u013.size();
for(int i=1;i<n13-1;i++) u013[i]=(-1)*u013[i];
lambda6=u013;

GlobalError13=frontera13.get_error1D(ErrInd13, u013, fe_U1,
                                     fe_U3);

cout << " ERROR GLOBAL ESTIMADO en la frontera 13= "
                                  << GlobalError13 << endl;

//
// Problema en la frontera interna 34
//

int numel34=TH34.get_numel();
int neq34=TH34.get_numnp();
vect_d ErrInd34(numel34);
```

```
// Declara un tipo de problema en la frontera 34
problem1d frontera34(&TH34, &fe1d_3, &flow);

cout << " Num de ecuaciones del sistema 34 = "<< neq34 <<endl;
vect_d b034(neq34);// aqui almacenamos el segundo miembro
vect_d u034(neq34);// aqui almacenamos la solucion del problema
frontera34.set_problem_second_member(fe_U3, fe_U4, b034);

// Calcula el error del algoritmo de Uzawa
double error34=0.;
for(int i=0;i<b034.size();i++) error34+=b034[i]*b034[i];
cout << " error UZAWA 34= "<< sqrt(error34)<<endl;
for(int i=0;i<b034.size();i++) b034[i]=rho*b034[i];
frontera34.add_second_member(lambda3, b034,0);
frontera34.add_second_member(lambda3, b034,1);

// Construye el sistema de ecuaciones con condiciones
// Dirichlet donde b es el segundo miembro del sistema
A34=frontera34.set_problem_matriz(b034,ua,ub);
A34->solve(b034,u034);
lambda3=u034;
int n34=u034.size();
for(int i=1;i<n34-1;i++) u034[i]=(-1)*u034[i];
lambda4=u034;
```

```
GlobalError34= frontera34.get_error1D(ErrInd34, u034, fe_U3,
                                      fe_U4);

cout << " ERROR GLOBAL ESTIMADO en la frontera 34= "
                                    << GlobalError34 << endl;

//
// Problema en la fronteira interior 24
//

int numel24=TH24.get_numel();
int neq24=TH24.get_numnp();
vect_d ErrInd24(numel24);

// Declara un tipo de problema en la frontera 24
problem1d frontera24(&TH24, &fe1d_4, &flow);

cout << " Num de ecuaciones del sistema 5 = "<< neq24 <<endl;
vect_d b024(neq24);// aqui almacenamos el segundo miembro
vect_d u024(neq24);// aqui almacenamos la solucion del problema
frontera24.set_problem_second_member(fe_U2, fe_U4, b024);

//Calcula el error del algoritmo de Uzawa
double error24=0.;
```

```
    for(int i=0;i<b024.size();i++) error24+=b024[i]*b024[i];
    cout << " error UZAWA 24= "<< sqrt(error24)<<endl;
    for(int i=0;i<b024.size();i++) b024[i]=rho*b024[i];
    frontera24.add_second_member(lambda3, b024,0);
    frontera24.add_second_member(lambda3, b024,1);

    // Construye el sistema de ecuaciones con condiciones
    //Dirichlet donde b es el segundo miembro del sistema
    A24=frontera24.set_problem_matriz(b024,ua,ub);
    A24->solve(b024,u024);

    lambda7=u024;
    int n24=u024.size();
    for(int i=1;i<n24-1;i++) u024[i]=(-1)*u024[i];
    lambda8=u024;

    GlobalError24= frontera24.get_error1D(ErrInd24, u024, fe_U2,
                                          fe_U4);
    cout << " ERROR GLOBAL ESTIMADO en la frontera= "
                                       << GlobalError24 << endl;
    }

  // Anoto el instante t2 y cuento los segundos transcurridos
  // desde que anote el instante t1
  t2 = times(&crono);
```

```
  double clk_tck =sysconf(_SC_CLK_TCK);  // Tics por segundo
  double t_real = (double)((t2 - t1) / clk_tck);
  double t_user =
            (double) ((crono.tms_utime+crono.tms_stime)/clk_tck);
  cout << "Tiempo Wall Clock real: " << t_real<< " seg."<< endl;
  cout << "Tiempo Wall Clock user: " << t_user << " seg."<< endl;
  cout << "Tiempo Wall Clock sys: " << t_real - t_user
                                              << " seg."<< endl;
  #ifdef DIBUJAR_FINAL
   fe_U1.plot();
   fe_U2.plot();
   fe_U3.plot();
   fe_U4.plot();
  #endif

  cout << "Programa terminado normalmente" << endl;

  return 0;

}
```

Capítulo 7

Conclusiones

En este capítulo, presentamos las principales conclusiones de este trabajo, así como también los problemas abiertos y futuras líneas de investigación.

Hemos propuesto un algoritmo adaptativo AMUADD basado en una iteración tipo Uzawa que mejora la lentitud del método Uzawa clásico. La adaptación del mallado hecha de forma independiente en cada subdominio mejora substancialmente la convergencia del algoritmo en comparación con el método de Uzawa clásico. En concreto podemos concluir:

- El algoritmo diseñado es robusto y combina técnicas de descomposición de dominios, donde las mallas se adaptan de forma independiente en cada subdominio, de acuerdo a las características especificas de la solución del problema en cada uno de ellos.

- El uso de un estimador a-posteriori basado en la estimación del residuo nos ha permitido obtener fiabilidad y eficiencia.

- Se ha obtenido la convergencia del AMUADD incluida la de los multiplicadores y demostramos la convergencia de la solución numérica.

- Las experiencias numéricas comprueban los resultados del análisis teórico así como la robustez del algoritmo.
- El AMUADD captura muy bien las capas límites, incluso en el caso en que la frontera entre dos subdominios dominios cruce la capa límite y los órdenes de convergencia obtenidos son óptimos.
- El AMUADD es indicado para ser usado en máquinas paralelas, ya que la solución del problema se calcula de forma independiente en cada subdominio, ahorrándose tiempo de cálculo y recursos del ordenador.

Problemas Abiertos

Nuestro algoritmo adaptativo ofrece extensiones y modificaciones, como por ejemplo:

- Refinar adaptativamente las mallas en las fronteras internas.
- Establecer nuevas estrategias de marcado.
- Extender el algoritmo a problemas de evolución.
- Mejorar la estimación a-posteriori del error y el critério de refinamiento para evitar el exceso de refinamiento en subdominios donde no hay "dificultades".

Capítulo 8

Publicaciones

A continuación se muestran las publicaciones en Congresos producidas durante el desarrollo de esta Tesis.

Publicaciones en Congresos

- *Métodos de Descomposición de Dominio con Adaptación de Mallado en Problemas de Convección-Reacción-Difusión.*
  M. Simões, L. Ferragut
  CMNE/CILAMCE 2007, Oporto (Portugal), 13-15 Junio, 2007.

- *Análisis de la Convergencia del M.E.F. en Algoritmos de Descomposición de Dominio con Adaptación de Mallado.*
  M. Simões, L. Ferragut.
  XX Congreso de Ecuaciones Diferenciales y Aplicaciones, X Congreso de Matemática Aplicada, Sevilla (España), 24-28 Septiembre, 2007.

- *A Numerical Adaptive Algorithm Combining Domain Decomposition.*
  M. Simões, L. Ferragut.

8th.World Congress on Computational Mechanics (WCCM8) 5th European Congress on Computational Methods in Applied Sciences and Engineeering (ECCOMAS 2008), Venice (Italy), June 30 - July 5, 2008.

Bibliografía

[1] F. ANDRÉS. *Métodos adaptativos para problemas no lineales asociados a operadores multívocos y aplicaciones.* PhD thesis, Facultad de Ciencias. Departamento de Matemáticas Aplicada.Universidad de Salamanca, 2005.

[2] I. BABUŠKA and W. C. RHEINBOLDT. Error estimates for adaptive finite element computations. *SIAM J. Numer. Anal.*, 15:736–754, 1978.

[3] I. BABUŠKA and W. C. RHEINBOLDT. A posterior error estimators in the finite element method. *Internat. J. Numer. Meth. Engrg.*, (12):1597–1615, 1978.

[4] I. BABUŠKA and T. STROUBOULIS. *The finite element method and its reability.* Numerical Mathematics and Scientific Computations, Clarendon Press-Oxford University Press, New York, 2001.

[5] N. BALIN, A. BENDALI, M. B. FARES, F. MILLOT, and N. ZERBIB. Some applications of substructuring and domain decomposition techniques to radiation and scattering of time-harmonic electromagnetic waves. *Comptes Rendus Physique*, 7(5):474–485, 2006.

[6] E. BÄNSCH, P. MORIN, and R. H. NOCHETTO. An adaptive uzawa fem for the stokes problem: Convergence without the inf-sup condition. *SIAM J. Numer. Anal.*, 40(4):1207–1229, 2002.

[7] Y. BOUBENDIR and A. BENDALI. Domain decomposition method for solving scattering problems by a boundary element method. *CIMNE: Barcelona 2002. Proceedings of the 13th International Conference on Domain Decomposition Methods, Lyon, France*, pages 319–326.

[8] Y. BOUBENDIR, A. BENDALI, and M. B. FARES. Coupling of a non-overlapping domain decomposition method for a nodal finit element method with a boundary element method. *Internat. J. Numer Meth. Engng.*, (73):1624–1650, 2008.

[9] F. BREZZI and M. FORTIN. *Mixed and Hybrid Finite Element Methods.* Springer Series in Computational Maths 15. Springer-Verlag, 1991.

[10] R. CHANDRA, R.MENON, L. DAGUM, D. KOHR, D. MAYDAN, and J.MCDONALD. *Parallel Programming in OpenMP.* Academic Press, 2001.

[11] B. CHAPMAN, G. JOST, and R. D. Pas. *Using OpenMP:Portable Shared Memory Parallel Programming (Scientific and Engineering Computation).* The MIT Press, 2007.

[12] P. G. CIARLET. *The Finite Element Method for Elliptic Problems.* North-Holland, Amsterdam, 1978.

[13] CLÉMENT. Approximation by finite element functions using local regularization. *RAIRO Anal. Numer.*, 9(R-2):77–84, 1975.

[14] F. COLLINO, S. GHANEMI, and P. JOLY. Domain decomposition method for harmonic wave propagation: a general presentation. *Computer Methods in Applied Mechanics and Engineering*, (184):171–211, 2000.

[15] B. DESPRÉS, P.JOLY, and J. ROBERTS. A domain decomposition method for the harmonic maxwell´s equations. *In IMACS International Symposium on Iterative Methods in Linear Algebra, Beauvens R, de Groen P (eds).North-Holland: Amesterdam*, pages 475–484, 1992.

[16] W. DÖFLER. A robust adaptive strategy for the nonlinear poisson equation. *Computing*, 55:289–304, 1995.

[17] W. DÖFLER. A convergent adaptive algorithm for poisson‘s equation. *SIAM J. Numer. Anal.*, 33(3):1106–1124, 1996.

[18] J. J. DONGARRA. Solving linear systems on vector and shared memory computers. *SIAM, Philadelphia, PA*, 1991.

[19] D. ELIAS. Introduction to parallel programming concepts. Cornell Theory Center, 1995. Workshop on Parallel Programming on the IBM SP. http://www.tc.cornell.edu/Edu/workshop.

[20] B. ERSLAND and M. ESPEDAL. On object oriented programming languages as a tool for a domain decomposition method with local adaptive refinement. *Ninth International Conference on Domain Decomposition Methods. Editor Petter E. Bjørstad, Magne S. Espedal and David E. Keyes*, 1998.

[21] L. FERRAGUT. Neptuno++. http://web.usal.es/ ∼ ferragut/.

[22] P. FREY. *An interative mesh visualization software.* 2001. http://www-rocq.inria.fr/gamma/medit/medit.html.

[23] R. GLOWINSKI. *Numerical Methods for Nonlinear Varational Problems.* Springer, New York, 1984.

[24] R. GLOWINSKI. *Numerical Methods for Fluids*, volume IX. P.G.Ciarlet y J. L. Lions Editors, 1th. Edition, 2003.

[25] R. GLOWINSKI and M. FORTIN. *Augmented Lagrangian methods: Applications to the numerical solution of boundary value problems*, volume 15. North Holland, Studies in Mathematics and its Applications. Amsterdam, 1983.

[26] R. GLOWINSKI, G.H. GOLUB, G.A. MEURANT, and J. PÉRIAUX eds. Domain decomposition methods for partial differential equations. *SIAM, Philadelphia, 1988. Proceedings of the First International Symposium on Domain Decomposition Methods for Partial Differential Equations, Paris, France, January 14-16.*

[27] G. H. GOLUB and C. F. VAN LOAN. *Matrix Computations.* 3rd edition, The Johns Hopkins University Press, Baltimore, 1996.

[28] W. D. GROPP and D.E. KEYES. Domain decomposition with local refinement. *SIAM J. Sci. Statist.*, 13:967–993, 1992.

[29] F. HECHT, O. PIRONNEAU, A. LE HYARÍC, and K. OHTSUKA. Freefem++ documentation, on the web at http://www.freefem.org/ff++.

[30] P. HIPTMAIR, B. WOHLMUTH, and T. SCHIEKOFER. Multilevel preconditioned augmented lagrangian techniques for 2nd order mixed problems. 2000.

[31] R. H. HOPPE and B. WOHLMUTH. Adaptive multilevel techniques for mixed finite element discretizations of elliptic boundary value problems. *SIAM J. Numer. Anal.*, 34(4):1658–1681, 1997.

[32] T. JAMIK. Parallel implementation of de p-version of the finite element method for elliptic equations on a shared-memory architecture. *Advances in Computational Mathematics*, (8):97–110, 1998.

[33] B. KÅGSTRÖM, E. ELMROTH, J. DONGARRA, and J. WAŚNIEWSKI. *Applied Parallel Computing: State of the Art in Scientific Computing. 8th International Workshop, PARA 2006, Umeå, Sweden, June 2006.* Springer.

[34] J. KRUIS. *Decomposition Methods for Distribued Computing.* Saxe Coburg Publications, 2006.

[35] V. KUMAR, A. GRAMA, A. GUPTA, and G. KARYPIS. *Introduction to Parallel Computing: Design and Analysis of Parallel Algoritms.* Benjamin/Cummings, Redwood City, USA, 1994.

[36] J. L. LIONS and O. PIRONNEAU. Domain decomposition methods for cad. *C. R. Acad. Sci. Paris*, t. 328(Sér. I):73–80, 1999.

[37] J. L. LIONS and O. PIRONNEAU. Non-overlapping domain decomposition for evolution operators. *C. R. Acad. Sci. Paris*, t. 330(Sér. I):943–950, 2000.

[38] J. L. LIONS and O. PIRONNEAU. Overlapping domain decomposition for evolution operators. *C. R. Acad. Sci. Paris*, t. 330(Sér. I):1–6, 2000.

[39] J. L. LIONS and O. PIRONNEAU. Virtual control, replicas and decompositon of operators. *C. R. Acad. Sci. Paris*, t. 330(Sér. I):47–54, 2000.

[40] R. I. MACKIE. *Object Oriented Methods and Finit Element Analysis*. Saxe Coburg Publications, 2001.

[41] A. MAROWKA. *Algorithms and Architectures for Parallel Processing: Performance of OpenMP Benchmarks on Multicore Processors*, volume 5022. Lecture Notes in Computer Science, Springer Berlin, 2008.

[42] P. MORIN, R. H. NOCHETTO, and K. G. SIEBERT. Data oscillation and convergence of adaptive fem. *SIAM J. Numer. Anal.*, 38(2):466–488, 2000.

[43] P. MORIN, R. H. NOCHETTO, and K. G. SIEBERT. Convergence of adaptive finite element methods. *SIAM Rev.*, 44(4):631–658, 2002.

[44] A. QUARTERONI. Domain decomposition and parallel processing for numerical solution of partial differential equations. *Surveys Math. Indust.*, (1):75–118, 1991.

[45] M. J. QUINN. *Parallel Programming in C with MPI and OpenMP*. McGraw-Hill, 2004.

[46] P. A. RAVIART and J. M. THOMAS. *Introduction à l´ Analyse Numérique des Equations aux Dérivées Partielles*. Masson, Paris, 1983.

[47] M. C. RIVARA. Algorithms for refining triangular grids suitable for adaptive and multigrid techniques. *Internat. J. Numer. Meth. Engrg.*, (20):745–756, 1984.

[48] M. C. RIVARA and G. IRIBRREN. The 4-triangles longest-side partition of triangles and linear refinement algorithms. *Math. Comput.*, 65(216):1485–1502, 1996.

[49] C. RIVERA, M. HENICHE, R. GLOWINSKI, and P. TANGUY. Parallel finite element simulations of incompressible viscous fluid flow by domain decomposition with lagrange multipliers. *Journal of Computational Physics*, 229(13):5123–5143, 2010.

[50] J. E. ROBERTS and J. M. THOMAS. *Mixed and Hybrid Methods, Handbood of Numerical Analysis, vol II. Finite Element Method (Part 1)*. Editado por P. G. Ciarlet y J. L. Lions. Elsevier Science Publishers B. V., 1991.

[51] J. SIMONS, S. SHAH, and K. Hotta. Openmp documentation, on the web at http://openmp.org.

[52] P. L. TALLEC. Domain decomposition methods in computational mechanics. *Computational Mechanics Advances*, 1994.

[53] R. VERFÜRTH. A posteriori error estimation and adaptive mesh-refinement techniques. *J. Comp. Appl. Math.*, 50:67–83, 1994.

[54] R. VERFÜRTH. *A review of a posteriori error estimation and adaptive mesh-refinement techniques*. Wiley-Teubner, 1995.