AMGLIB++: LIBRERÍA ORIENTADA A OBJETOS PARA RESOLVER GRANDES SISTEMAS LINEALES ESPARCIDOS BASADA EN EL MÉTODO DE MULTIMALLA ALGEBRAICO

Por

Yurin Holguino Borda

Tesis sometida en cumplimiento parcial de los requerimientos para el grado de

MAESTRÍA EN CIENCIAS

en

COMPUTACIÓN CIENTÍFICA

UNIVERSIDAD DE PUERTO RICO
RECINTO UNIVERSITARIO DE MAYAGÜEZ

Mayo, 2006

Aprobada por:

Krzysztof Rozga, Ph.D. Miembro, Comité Graduado	Fecha
Robert Acar, Ph.D. Miembro, Comité Graduado	Fecha
Paul Castillo, Ph.D. Presidente, Comité Graduado	Fecha
Esov Velázquez, Ph.D. Representante de Estudios Graduados	Fecha
Luis Cáceres, Ph.D. Director del Departmento	Fecha

Abstract of Disertación Presented to the Graduate School
of the University of Puerto Rico in Partial Fulfillment of the
Requirements for the Degree of Master of Science

AMGLIB++: AN OBJECT ORIENTED LIBRARY FOR SOLVING LARGE SPARSE LINEAR SYSTEMS BASED ON ALGEBRAIC MULTIGRID METHOD

By

Yurin Holguino Borda

May 2006

Chair: Paul Castillo, Ph.D.
Major Department: Mathematics

Numerical solutions of scientific and engineering applications often requiere the solution of large sparse linear systems which arise from discretizations of Partial Diferential Equations. Under this motivation, in this work we build a library based on Algebraic Multigrid method called AMGLib++. The library is implemented as a *Computational Abstract Framework*. It brings flexibility and high customization for modifying the source code. Throughout the work an object oriented methodology and generic programming paradigm are employed.

Resumen de Disertación Presentado a Escuela Graduada
de la Universidad de Puerto Rico como requisito parcial de los
Requerimientos para el grado de Maestría en Ciencias

AMGLIB++: LIBRERÍA ORIENTADA A OBJETOS PARA RESOLVER GRANDES SISTEMAS LINEALES ESPARCIDOS BASADA EN EL MÉTODO DE MULTIMALLA ALGEBRAICO

Por

Yurin Holguino Borda

Mayo 2006

Consejero: Paul Castillo, Ph.D.
Departamento: Matemáticas

Las simulaciones numéricas de aplicaciones científicas y de ingeniería generalmente requieren resolver grandes sistemas esparcidos, provenientes de la discretización de ecuaciones diferenciales parciales. Bajo esta motivación, en el presente trabajo construimos una librería basada en el método multimalla algebraico denominada AMGLib++, la cual es concebida como un *Entorno Computacional Abstracto*. La librería brinda alta personalización en el manejo del código. Está construida bajo la metodología orientada a objetos y el paradigma de la programación genérica.

A mis padres Julia y Félix por su apoyo y estímulo en todo momento. A mis hermanos Max, Elízabeth, Ernesto, Alex, Héctor, Carlos, Jonathan y Rubén a quienes siempre los tengo presentes y a todos mis amigos y compañeros.

AGRADECIMIENTOS

A mi asesor profesor Paul Castillo, quien me guió a lo largo del desarrollo del presente trabajo así como en la duración de mis estudios en la maestría.

TABLA DE CONTENIDO

pagina

LISTA DE TABLAS

LISTA DE FIGURAS

LISTA DE ABREVIATURAS

EDP	Ecuación diferencial parcial.
MMG	Multimalla geométrico.
MMA	Multimalla algebraico.

LISTA DE SIMBOLOS

u	Vector Solucion.
f	Lado derecho del sistema lineal.
A	Matriz del sistema lineal.
A_H	Matriz del sistema lineal en el nivel grueso.
A_h	Matriz del sistema lineal en el nivel fino.
u_H	Vector solución del sistema lineal en el nivel grueso.
u_h	Vector solución del sistema lineal en el nivel fino.
f_H	Lado derecho del sistema lineal en el nivel grueso.
f_h	Lado derecho del sistema lineal en el nivel grueso.
C	Variable gruesa.
F	Variable fina.
P	Operator de interpolación.
R	Operator de restricción.

CAPITULO 1
INTRODUCCIÓN

Las simulaciones numéricas de aplicaciones científicas y de ingeniería generalmente requieren resolver grandes sistemas esparcidos, provenientes de la discretización de una Ecuación Diferencial Parcial (EDP) o un sistema de ecuaciones diferenciales parciales. Bajo esta motivación en el presente trabajo construimos una librería basada en el método multimalla algebraico, para resolver numéricamente EDPs, que sea óptima en tiempo y espacio, además, que el usuario pueda modificar, personalizar o adaptar según la aplicación o discretización.

Los sistemas de ecuaciones que son generados en la discretización de EDPs fácilmente llegan a ser del orden de miles o millones de incógnitas y se incrementa aún más en un problema definido en tres dimensiones, donde el cuello de botella precisamente llega a ser la resolución del sistema de ecuaciones. Los métodos multimallas estan orientados a superar esta complejidad ofreciendo la posibilidad de resolver en orden de $O(n)$ donde n representa el número de incógnitas.

Los métodos de multimalla son métodos iterativos eficientes para resolver sistemas lineales de ecuaciones, que a partir de la malla inicial, denominada *Malla Fina*, generará múltiples mallas, conocidas como *Mallas Gruesas*, para reducir uniformemente el error, independientemente del tamaño del problema, es decir, que el número de iteraciones no varía mucho si el número de variables en el sistema de ecuaciones (la matriz del sistema lineal) se incrementa, una característica muy

notable de los métodos de multimalla.

El multimalla algebraico tiene sus bases sobre su antecesor: el *multimalla geométrico*, extendiendo sus ideas clásicas, pero sin tomar en cuenta la geometría del dominio de la EDP [49, 50] y de ahí precisamente su nombre, ya que toma un enfoque puramente basado en la matriz, o dicho de otro modo, en el sistema *algebraico* de ecuaciones. Aunque el multimalla algebraico es una versión reciente de los métodos multimalla, no significa que sea un competidor de su homólogo geométrico, sino que es una alternativa en problemas que se ajusten al mismo.

Las investigaciones y aplicaciones del multimalla algebraico han tenido un gran impacto en los últimos años. La contribución en eficiencia espacial como temporal resulta ser muy atractiva en el desarrollo de software númerico, las aplicaciones cada vez son mayores, más si su desempeño con problemas donde la geometría del dominio es compleja, el multimalla algebraico se convierte en una gran opción.

En la actualidad existe gran variedad de librerías que implementan multimalla algebraico, ya sea de forma general o con algún enfoque particular, entre ellos se puede mencionar a:

1.1 hypre ("High Performance Preconditioner")

Desarrollada en *Lawrence Livermore National Laboratory*, una librería que utiliza cómputo masivo en paralelo, contiene la implementación de varios precondicionadores multinivel [23, 25]. Entre las bondades de esta librería se tiene el uso de interfaces conceptuales [24], los cuales estan relacionados a la capacidad de permitir al usuario acceso a la librería de forma compatible y natural a como está definido un problema, es decir, si fue definido sobre mallas estructuradas, no estructuradas,

de diferencias finitas, elementos finitos, etc. *hypre* brinda un acceso especializado a cada uno de estos.

Hypre provee precondicionadores escalables y métodos basados en Krylov, soporte para matrices esparcidas. *hypre* trae consigo también SMG (semi coarsening multigrid), para resolver sistemas lineales construidos a partir de una discretización de la ecuación de difusión, de la forma: $\nabla \cdot (D\nabla u) + \sigma u = f$. Con D conteniendo los coeficientes de difusión, $\sigma \geq 0$; definido sobre un dominio rectangular, en dos dimensiones, con 9 puntos de esténcil y hasta de 27 puntos de esténcil en tres dimensiones, las discretizaciones pueden provenir de diferencias finitas, volumen finito o elementos finitos.

La forma de realizar el suavizado, en el caso de tres dimensiones, es primero por planos para luego pasar a un suavizado por líneas, este último aplicado también al caso de dos dimensiones. Un componente similar al SMG, que es incluido en *hypre*, es el PFMG ("Parallel Semicoarsening multigrid solver"), con ligeras modificaciones del anterior, se diferencia principalmente en la eficiencia de la implementación del algoritmo del multimalla.

Otro componente importante de *hypre* es BoomerAMG [56], que implementa el mutimalla algebraico, el cual puede ser usado como método de solución independiente o como precondicionador. BoomerAMG contiene varias estrategias de selección de malla gruesa y suavizadores.

ParaSails [18, 20] ("Parallel Sparse Approximate Inverse (Least-Squares) Preconditioner"), un componente incluido en *hypre*, es una implementación en paralelo

de un precondicionador como inversa aproximada, emplea un patrón de esparcidad *a priori* [19], utilizando una técnica de *post-filtrado*, para reducir el costo en la aplicación del precondicionamiento. También esta Euclid, con factorizaciones incompletas tipo ILU en paralelo y precondicionado basado en el algoritmo planteado por Yousef Saad.

Por otro lado esta librería permite elegir entre modo de depuración y modo optimizado, hace uso de las librerías BLAS (véase apéndice) y en cuanto a computación en paralelo, está basado en *MPI* ("Message Passing Interface") con opción de incluir hilos de ejecución con *OpenMP*, pero solo para algunos métodos de solución, está escrito en C, con interfaces tanto para C y para FORTRAN.

1.2 ML ("Multilevel Preconditioning Package")

Es una librería para multimalla como precondicionador en paralelo [51], contiene varias implementaciones de MMA con estrategias de selección de mallas gruesas como son el clásico y el basado en agregación, incluye algunas variantes en la implementación de suavizadores (véase capítulo 3 de multimalla algebraico) basados en Gauss-Seidel y una versión especial para ecuaciones de Maxwell, esta librería es desarrollada en Sandia National Laboratories.

ML está diseñado para operar en conjunto con AztecOO, un componente de software orientado a objetos, para resolver sistemas lineales. Este paquete a su vez forma parte de la librería denominada *Trilinos*, también desarrollada en Sandia National Laboratories.

1.3 pARMS ("Parallel Algebraic Multilevel Solver")

Es una versión en paralelo de ARMS [48], no es directamente un multimalla algebraico, pero tiene un enfoque recursivo y multinivel. Construye el precondicionador dividiendo en dos bloques de variables gruesas y finas, con el fin de formar un complemento de Schur, que está relacionado con las variables gruesas, esto conduce a su vez a una forma natural de factorización LU, que se usa como precondicionador para el sistema, el cual es resuelto recursivamente aplicando el mismo criterio.

El reordenamiento de las variables es equivalente al realizado en el contexto del multimalla (denominado agregación), con este reordenamiento se busca agrupar variables que esten acopladas o relacionadas. Si tiene la matriz A, en el nivel l se puede expresar esta como:

$$P_l A_l P l^T = \begin{pmatrix} B_l & F_l \\ E_l & C_l \end{pmatrix}$$

Donde P_l es una matriz de permutación en el nivel l, cuando $l = 0$, A_0 es la matriz original. Así se puede representar como:

$$P_l A_l P l^T \approx \begin{pmatrix} L_l & 0 \\ E_l U_l^{-1} & I \end{pmatrix} \times \begin{pmatrix} U_l & L_l^{-1} F_l \\ 0 & A_{l+1} \end{pmatrix}$$

Donde A_{l+1}, es una aproximación del complemento Schur con respecto a C_l, de la forma:

$$A_{l+1} \approx C_l - (E_l U_l^{-1})(L_l^{-1} F_l)$$

En la versión en paralelo se toma un enfoque distribuido de matrices esparcidas para los bloques diagonales del complemento de Schur [47]. Esta librería está

desarrollada por Yousef Saad [37] en la Universidad de Minnesota, esta escrito en C y utiliza MPI para procesamiento en paralelo.

1.4 Prometheus

Construida por Mark Adams et al. [5], es otra librería que implementa tanto un multimalla geométrico como un algebraico, este último denominado ATLAS, el cual está basado en agregación suavizada, un algoritmo de selección de malla gruesa introducido por Vanek et al. [54], relacionado a la forma de construir el operador de interpolación, que puede ser construido en dos pasos, formando primero los agregados (grupos) luego construyendo un operador de interpolación tentativo al cual se aplica un suavizador. El operador de interpolación es construido dependiendo de cada aplicación, un ejemplo podría ser asignar un valor de uno en la entrada (i, j), si la variable i está contenida en el agregado j y asignar cero en caso contrario. Para el suavizado se puede aplicar cualquier método iterativo clásico, por ejemplo si consideramos el método de ω-Jacobi, de la forma: $P = (I - \omega D^{-1}A)\tilde{\mathrm{P}}$, que representa a la la iteración de ω-Jacobi, con A la matriz del sistema, D la diagonal de A y $\tilde{\mathrm{P}}$, el operador de interpolación formado tentativamente. Ésto se aplica a cada nivel, en la fase de configuración del multimalla.

Esta librería está orientada a resolver sistemas lineales esparcidos obtenidos de discretizaciones por elementos finitos en tres dimensiones, está construida en C++ con enlaces para C,FORTRAN y soporte de computación en paralelo.

Las librerías citadas anteriormente están disponibles libremente, lo que no es el caso de otras librerías tales como LAMG[2] (Los Alamos Algebraic Multigrid Code) desarrollado en Los Alamos National Laboratories, basado en una selección agresiva de malla gruesa, está codificado en FORTRAN 95. PEBBLES [4] para

sistemas provenientes de ecuaciones elípticas, de segundo orden, desarrollado por la Universidad de Linz, con varias estrategias de interpolación y selección de malla gruesa. SAMG [1], librería escrita en FORTRAN 90, en versión no paralela y paralela (SAMGp), permite varios tipos de selección de mallas gruesas, y un manejo eficiente de memoria, desarrollado por el Instituto de Investigación Fraunhofer en Algoritmos y Computación Científica. Información de otras librerías y/o códigos se puede encontrar en la página web de MGNet [3], que contiene más información relacionada con métodos multimallas.

A pesar de las bondades que exhiben las anteriores librerías, la modificación y personalización de código demanda un conocimiento profundo de las rutinas y tipos de datos que cada librería define, frente a ésto, en el presente trabajo se plantea una nueva librería: AMGLib++ concebida como un *Entorno Abstracto*, cuya principal característica es precisamente brindar alta personalización en el manejo del código para el multimalla algebraico. Ademas de ello, podemos mencionar otras características tales como:

1. Código abierto, porque bajo esta filosofía, el código fuente estará disponible libremente para su modificación y distribución.
2. Por otro lado, se utiliza la metodología orientada a objetos, donde los componentes del software reflejan entidades del dominio del problema, de forma natural y mutuamente compatibles, entre otras ventajas que son útiles de la metodología orientada a objetos, tanto para la programación como para el mantenimiento del código.
3. Otra característica importante es el uso del paradigma de la *Programación Genérica*, para brindar mayor abstracción de este modo proveer una gran flexibilidad, extensibilidad y alta personalización. La idea es brindar un entorno en el cual añadir

y/o modificar componentes del multigrid algebraico sea fácil, por ejemplo si se tiene una clase para manejar matrices en un formato particular, solo será necesario cuidar que existan los métodos o funciones necesarios para interactuar con AMGLib++.

En la figura 1–1 se muestra la idea de esta librería.

Figura 1–1: AMGLib++: Ensamblamiento de componentes en la librería.

La organización del presente trabajo es de la siguiente forma: En el capítulo II se mostrarán las nociones básicas de los métodos iterativos clásicos, estacionarios y no estacionarios.

Las ideas básicas sobre el método de multimalla, obviamente con énfasis en los multimalla algebraicos, la historia y evolución, su predecesor, el multimalla

geométrico, las diferencias entre ambos y otros aspectos se trataran en el capítulo III.

En el capítulo IV haremos referencia a algunas formas más usuales de la representación de matrices esparcidas.

Sobre la programación genérica o politípica se discutirá en el capítulo V, la base de este paradigma así como las bondades que aporta su utilización.

En el capítulo VI, mostraremos el entorno abstracto, de como se realiza la adaptación del multimalla algebraico versus plantillas de clases y aspectos relacionados con su implementación, es decir la abstracción matemática.

Sobre los experimentos realizados con esta librería y la forma en la cual se utiliza, los diferentes componentes se mostrará en el capítulo VII, con los ejemplos respectivos.

CAPITULO 2
MÉTODOS ITERATIVOS

Un sistema lineal de ecuaciones la podemos representar de la forma:

$$Ax = b, \tag{2.1}$$

donde A es una matriz no singular de dimensión $n \times n$, x y b vectores de tamaño n. Los métodos para resolver un sistema de la forma (2.1) se pueden clasificar en métodos directos e iterativos. Los primeros son aquellos que determinan la solución en un número determinado de pasos, teóricamente se obtiene una solución exacta, pero computacionalmente se puede complicar debido a los errores de redondeo. Entre los métodos directos se puede mencionar a la eliminación Gaussiana, factorizaciones como Cholesky, LU, QR, etc. Los métodos iterativos por otra parte, construyen la solución a partir de una serie de soluciones aproximadas. Detalles teóricos y prácticos sobre métodos directos así como métodos iterativos se puede encontrar en [9, 27, 29, 32, 46, 52, 55].

Una forma de realizar los métodos iterativos es considerar a la matriz A como $A = M - N$, donde M y N son dos matrices de la misma dimensión de A, con M no singular, reemplazando en (2.1), se tiene:

$$(M - N)x = b \tag{2.2}$$

$$Mx = Nx + b \tag{2.3}$$

Partiendo de un valor inicial $x^{(0)}$, se tiene el esquema de iteración general:

$$Mx^{(i+1)} = Nx^{(i)} + b, \tag{2.4}$$

con $i = 0, 1, 2, ...$, bajo ciertas condiciones la sucesión $\{x^{(i)}\}$ converge a la solución del sistema (2.1), si la convergencia es rápida el proceso de resolver el sistema se completará en pocos pasos.

Definición 2.0.1. *Sea $T = M^{-1}N$, donde $A = M - N$, a la matriz T se le denomina matriz de iteración.*

En la i-ésima iteración los errores estaran dados por: $e^{(i)} = x^{(i)} - x$, con $i = 0, 1, 2, ...$, restando (2.3) de (2.4) y mulplicando por $M^{-1}N$.

$$e^{(i)} = Te^{(i-1)} \tag{2.5}$$

$$e^{(i)} = T^2e^{(i-2)} \tag{2.6}$$

$$e^{(i)} = T^{(i)}e^{(0)}, \tag{2.7}$$

que representa el error en la i-ésima iteración, con respecto al error inicial.
Por otra parte, la matriz de iteración T, puede ser escrito como: $T = I - M^{-1}A$. De (2.4), podemos escribir:

$$x^{(i+1)} = M^{-1}Nx^{(i)} + M^{-1}b \tag{2.8}$$

$$x^{(i+1)} = Tx^{(i)} + c, \tag{2.9}$$

donde $c = M^{-1}b$. estos métodos son conocidos como estacionarios, porque la transición desde $x^{(i)}$ hasta $x^{(i+1)}$ depende unicamente del valor anterior y no de toda la historia anterior, es decir: $x^{(i+1)} = f(x^{(i)})$, una función de la iteración anterior. Entre estos métodos están Richardson, Jacobi, Gauss-Seidel, etc. Por otra parte, si $x^{(i+1)} = f(x^{(i)}, x^{(i-1)}, ...)$, son llamados métodos no estacionarios, como el método

de gradiente conjugado, GMRES, etc.

La ventaja de los métodos iterativos es la simplicidad, porque solo se realizarán multiplicaciones matriz-vector y adición de vectores, además de ello, si la matriz es esparcida (vease capítulo IV), como generalmente resulta de las discretizaciones provenientes de EPDs, se puede explotar esta característica. La desventaja de estos métodos es que pueden presentar una convergencia lenta, para lo cual será necesario añadir una condición adicional de parada o fin.

Definición 2.0.2. *Se llama residuo al vector r tal que, $r = b - Ax$, donde r, b, x son vectores de dimension n y A es una matriz de dimensión $n \times n$.*

Una condición de parada o fin en el proceso de resolver el sistema (2.1), se puede establecer como:

$$\frac{\| r^{(i)} \|}{\| r^{(0)} \|} \leq Tol$$

Para una tolerancia, Tol dada y donde $r^{(i)}$, es el residuo en la i-ésima iteración y $r^{(0)}$, el residuo inicial.

Definición 2.0.3. *Sea A una matriz, el espectro de A, denotado por σ, se define como: $\sigma = \{\lambda \in C \mid \lambda$ es un autovalor de $A\}$.*

Definición 2.0.4. *Si σ es el espectro de una matriz A, entonces $\rho(A) = \max_{\lambda \in \sigma(A)} \mid \lambda \mid$ es llamado el radio espectral de A.*

Analizando el radio espectral de la matriz de iteración se puede determinar la convergencia del método, ésto se expresa en el siguiente teorema.

Teorema 1. *Sea T una matriz de iteración, T es convergente si y solo si $\rho(T) < 1$*

Prueba. *Como T es la matriz de iteración, notese que $\rho(T) < 1 \Leftrightarrow (T)^k_{k\to\infty}, \to 0$, es decir:*

$\rho(T) < 1 \Leftrightarrow \lim_{n\to\infty} T^n = 0$

$(\Longrightarrow)$

Primero probemos que si $T^n = 0 \Rightarrow \rho(T) < 1$

Sea λ un valor propio de T, entonces existe $x \in \mathbb{C}, x \neq 0$

se tiene que: $Tx = \lambda x$

luego: $T^n x = \lambda^n x$

$\| T^n \| \| x \| \geq \| T^n x \| = \| \lambda^n x \| = |\lambda|^n \| x \|$

$\| T^n \| \geq |\lambda|^n$

Como $\| T^n \|$ tiende a cero entonces $|\lambda| \to 0$

Por otra parte se sabe que $|\lambda| < 1$ es cierto para cualquier valor propio λ, entonces $\rho(T) < 1$

$(\Longleftarrow)$,

$\rho(T) < 1 \to \lim_{n\to\infty} T^n = 0$

Utilizando la descomposición de Jordan, existe una matriz S no singular y una matriz de Jordan J, tales que:

$$SJS^{-1} = T$$

Donde:

$$J = \begin{pmatrix} J_1 & \dots & 0 \\ \vdots & J_2 & \\ & \ddots & \\ 0 & & J_p \end{pmatrix}$$

Así cada matriz J_k, de $n_k \times n_k$, con $1 \leq k \leq p$, será de la forma

$$J_k = \begin{pmatrix} \lambda_k & \dots & 0 \\ \vdots & \ddots & \\ & & \\ 0 & & \lambda_k \end{pmatrix} + \begin{pmatrix} 0 & 1 & \dots & 0 \\ \vdots & \ddots & \ddots & \\ & & & 1 \\ 0 & & & 0 \end{pmatrix}$$

Debido a esta especial distribución de entradas en las matrices J_k, para J_k^m, se tiene

$$J_k^m(i,j) = \begin{cases} 0, & si\ j \leq i \\ \binom{m}{j-i}\lambda_k^{m-j+i} & para\ i \leq j \leq min(n_k, m+i) \\ 0, & si\ m+i < j \leq n_k \end{cases}$$

Por otra parte $T^m = SJ^mS^{-1}$ y como $\rho(T) < 1$, entonces $|\lambda_k| < 1$, para todo $1 \leq k \leq p$, luego $\lim_{m\to\infty} J_k^m(i,j) = 0$, para todo $1 \leq i, j \leq n_k$.

Así cada J_k es convergente por consiguiente T también es convergente.

Definición 2.0.5. *Una matriz cuadrada A es estrictamente diagonal dominante si se cumple que: $\sum_{i\neq j}^{n} \mid a_{i,j} \mid < \mid a_{i,i} \mid$,*

donde n es la dimensión de la matriz A, además $i, j = 1, 2, \dots, n$.

2.1 Método de Jacobi

Si en (2.4) se hace $M = D$, la diagonal de la matriz A, entonces se tiene el método de Jacobi. Además las entradas de la diagonal deben ser distintas de cero.

$$x^{(i+1)} = x^{(i)} + D^{-1}(f - Ax^{(i)}) \tag{2.10}$$

Teorema 2. *Si A es una matriz estrictamente diagonal dominante, entonces la iteración de Jacobi converge para cualquier valor inicial x_0.*

Prueba. *De la matriz de iteración $M^{-1}N = D^{-1}(D - A) = I - D^{-1}A$*

$$= \begin{pmatrix} 1 & & 0 \\ & \ddots & \\ 0 & & 1 \end{pmatrix} - \begin{pmatrix} 1 & & \frac{a_{1n}}{a_{11}} \\ & \ddots & \\ \frac{a_{n1}}{a_{nn}} & & 1 \end{pmatrix}$$

$$= - \begin{pmatrix} 0 & \frac{a_{12}}{a_{11}} & \dots & \frac{a_{1n}}{a_{11}} \\ \frac{a_{21}}{a_{22}} & \ddots & \dots & \frac{a_{2n}}{a_{22}} \\ \vdots & & & \\ \frac{a_{n1}}{a_{nn}} & \dots & \frac{a_{nn-1}}{a_{nn}} & 0 \end{pmatrix}$$

Y como A es diagonalmente dominante (estrictamente), entonces se cumple que: $\| M^{-1}N \|_\infty < 1$, luego el método converge.

2.2 Método de Gauss-Seidel

En lugar de tomar solo la diagonal se tiene dos opciones tomar $M = D + L$ o $M = D + U$ donde L es la matriz triangular estrictamente inferior y U es la matriz triangular estrictamente superior de la matriz A. En el primer caso se tendrá, *"Forward Gauss-Seidel"* (2.11) y en el segundo *"Backward Gauss-Seidel"* (2.12).

$$x^{(i+1)} = x^{(i)} + (L + D)^{-1}(f - Ax^{(i)}) \tag{2.11}$$

$$x^{(i+1)} = x^{(i)} + (D+U)^{-1}(f - Ax^{(i)}), \tag{2.12}$$

Teorema 3. *Si A es una matriz estrictamente diagonal dominante, entonces la iteración de Gauss-Seidel converge para cualquier valor inicial x_0.*

Prueba. *Como $M = D+L, M = D+U$ y $A = M - N$, entonces se tiene la matriz de la iteracion como: $M^{-1}N = (L+D)^{-1}(-U)$*
Sea λ un valor propio de $M^{-1}N$ y $x_\lambda \in R^n$, con $x_\lambda \neq 0$, un vector propio asociado a λ, luego:

$$M^{-1}Nx_\lambda = \lambda x_\lambda$$

$$Nx_\lambda = \lambda M x_\lambda$$

$$-Ux_\lambda = \lambda(L+D)x_\lambda$$

Sin perdida de generalidad podemos escoger x_λ, de forma tal que $\| x_\lambda \|_\infty = 1$
Sea el indice i, tal que $x_i^\lambda = \| x_\lambda \|_\infty = 1$

$$-\sum_{j>i}^{n} a_{ij}x_j = \lambda \sum_{j\leq i}^{n} a_{ij}x_j$$

$$|\lambda| = \frac{|\sum_{j>i}^{n} a_{ij}x_i|}{|\sum_{j\leq i}^{n} a_{ij}x_j|}$$

$$\leq \frac{\sum_{j>i}^{n} |a_{ij}|}{|\sum_{j\leq i}^{n} a_{ij}x_j|}$$

$$\sum_{j\leq i}^{n} a_{ij}x_j = a_{ii}x_i + \sum_{j<i}^{n} a_{ij}x_i = a_{ii} + \sum_{j<i}^{n} a_{ij}x_j$$

Se puede notar que:

$$|a_{ii}| = |a_{ii} + \sum_{j<i}^{n} a_{ij}x_i - \sum_{j<i}^{n} a_{ij}x_i|$$

$$\leq |a_{ii} + \sum_{j<i}^{n} a_{ij}x_j| + \sum_{j<i}^{n} |a_{ij}|$$

Por ser diagonalmente dominate se cumple que:

$$0 < |a_{ii}| - \sum_{j<i}^{n} |a_{ij}| \leq |a_{ij} + \sum_{j<i}^{n} a_{ij}|$$

$$|\lambda| \leq \frac{\sum_{j>i}^{n} |a_{ij}|}{|a_{ii}| - \sum_{j<i}^{n} |a_{ij}|} < 1$$

Asi $\rho(M^{-1}N) < 1$

2.3 Método de Gradiente Conjugado

Es un método para resolver sistemas simétricos definidos positivos,donde a diferencia de los métodos anteriores no hay una matriz de iteración, construye la iterada $x^{(i)}$, como un elemento de:

$$x^{(0)} + span\{r^{(0)}, Ar^{(0)}, A^2r^{(0)}, A^3r^{(0)}, ..., A^kr^{(0)}\},$$

donde $r^{(0)} = b - Ax^{(0)}$, representa el residuo inicial.

Este método se basa en minimización de un funcional cuadrático convexo:

$$\phi(u) = \frac{1}{2}\langle u, u\rangle_A - \langle u, f\rangle \tag{2.13}$$

El funcional ϕ alcanza un mínimo precisamente cuando $Ax = f$. Mayores detalles prácticos así como teóricos de este método iterativo se puede encontrar en [9, 29, 46].

2.4 Gradiente Conjugado Precondicionado

La motivación de un precondicionamiento es la reducción del número de iteraciones, si se tiene el sistema original $Ax = f$ se transformará en uno equivalente:

$\tilde{A}x=\tilde{f}$ donde el espectro sea más favorable para la resolución de tal forma que resolver este nuevo sistema menos costoso que el original y por supuesto, que ambos sistemas tengan la misma solución. Tanto $\tilde{A}$ como $\tilde{f}$, se obtienen multiplicando ya sea por la derecha o por la izquierda con otra matriz P, llamada *Precondicionador*. Evidentemente el precondicionador no debe ser singular y debe garantizar que se preserve la simetría de la matriz original A.

El método del gradiente conjugado puede ser acelerado a través de un precondicionamiento, el cual se logra multiplicando por el precondicionador P^{-1}. Uno de los requerimientos importantes desde el punto de vista práctico es que resolver el sistema de la forma $Pz = y$, no sea costoso, porque si se observa en el algoritmo que se expondrá a continuación, el sistema $Mz_{k+1} = r_{k+1}$ se resuelve en cada paso de la iteracion del algoritmo.

El algoritmo del gradiente conjugado precondicionado es el siguiente:

1	$r_0 = f - Au_0$
2	$z_0 = M^{-1}r_0$
3	$p_0 = z_0$
4	for $k = 0$: MaxIter
5	{
6	$\alpha = \langle r_k, z_k\rangle / \langle Ap_k, p_k\rangle$
7	$u_{k+1} = u_k + \alpha_k p_k$
8	$r_{k+1} = r_k - \alpha_k Ap_k$
9	$z_{k+1} = M^{-1}r_{k+1}$
10	$\beta_k = \langle r_{k+1}, z_{k+1}\rangle / \langle r_k, z_k\rangle$
11	$p_{k+1} = z_{k+1} + \beta_k p_k$
12	}

Donde f, z, u, r p, u_0, r_0 son vectores de tamaño n, con u_0 el valor inicial, r_0 el residuo inicial, A es la matriz del sistema lineal, M el precondicionador, α y β escalares y $\langle , \rangle$ representa el producto interior de vectores.

Aspectos prácticos y de implementanción de todos estos métodos descritos anteriormente se puede obtener en [14, 46].

CAPITULO 3
MULTIMALLA

3.1 Referencias Históricas

En 1964 Fedorenko [26], realizó la primera publicación sobre el método de multimalla, con un esquema de discretización de diferencias finitas, con un esténcil de cinco puntos para resolver la ecuación de Laplace en un dominio rectangular. En 1966 Bachvalov extiende estas ideas a una ecuación más general, empleando coeficientes variables, en un esquema de diferencias centradas, con consideraciones teóricas más complejas. Pero todavía entonces no era considerado un método eficiente para ponerlo en práctica. Brandt [15] con una publicación inicial en 1973 y posteriormente en 1977 [16] mostró los principios y la utilidad práctica de los multimallas.

Por otra parte Hackbush [28] en 1976, también propone este método, exhibiendo detalles prácticos y teóricos, analizando la convergencia de estos métodos para ciertas EDPs.

En los años 80s numerosos investigadores realizan aportaciones significativas en el desarrollo de los métodos de multimalla, entre los que se destacan Brandt, McCormick y Ruge [17, 40], ampliando los horizontes para este método. Hoy en día se utiliza ampliamente este método y sus variantes, con una amplia gama de aplicaciones y áreas como economía, procesamiento de imágenes, geología, dinámica

de fluidos computacional, optimización, etc.

3.2 Multimalla Geométrico

La idea del multimalla surgió del análisis de los métodos iterativos clásicos tales como Gauss-Seidel, Jacobi, Richardson, etc. Se pudo notar que estos métodos reducen rápidamente las altas frecuencias de los componentes del error, pero lentamente las de baja frecuencia, por ello a este tipo de métodos se les suelen llamar también *Suavizadores*. Wesseling en [58] presenta un análisis detallado. Es importante subrayar esta caracteristica porque si de algún modo se usara estos mismos métodos para reducir rápidamente los componentes con bajas frecuencias entonces se tendría un método que en pocos pasos hallaria una solución aceptable al sistema lineal.

Si transferimos a otra malla más gruesa, los componentes cuyas frecuencias no pudieron ser reducidas, de forma tal que estas tomarán el comportamiento de componentes con altas frecuencias, se podría aplicar el mismo procedimiento empleado en la malla fina, reduciendo tanto las bajas como las altas frecuencias de los componentes del error o del residual.

En el contexto de multimalla, al proceso de reducir los componentes de error con frecuencias bajas, se le suele denominar *Corrección en Malla Gruesa*. En otras palabras componentes que no pudieron ser reducidos por los suavizadores serán reducidos por corrección en la malla gruesa, como resultado de esta combinación se tendrá una convergencia robusta y por consiguiente un método más eficiente, lo que precisamente refleja la idea de los métodos multimalla.

El nombre de multimalla sugiere la idea de emplear una serie de mallas gruesas, para la resolución del mismo problema, extendiendo a multiples mallas se reducirá rápidamente las oscilaciones de los componentes del error.

La transición entre la malla fina y una gruesa involucra una restricción y la transición de una más gruesa a una más fina se llama interpolación o prolongación. Con la restricción se transforma el residuo a la siguiente malla gruesa, para luego resolver, mientras que con la interpolación se envia el error obtenido en una malla gruesa a la malla inmediatamente más fina.

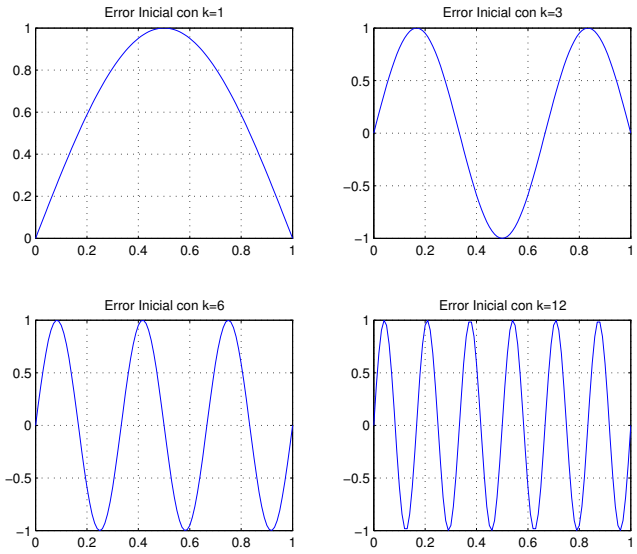

Figura 3–1: Error inicial, con modos de Fourier k=1,3,6,12

A continuación se muestra un ejemplo para ilustrar la idea. Consideremos la ecuación de Poisson $-\Delta u = f$ en $\Omega = [0,1]$, con condiciones de frontera: $u(0) = u(1) = 0$, para simplicidad asumamos un problema en una dimensión.

Discretizando mediante diferencias finitas, con la formula central, para el operador de segundo orden:

$$-\Delta u = -\frac{U_{i+1} - 2U_i + U_{i-1}}{h^2}$$

En notación de esténcil:

$$\begin{bmatrix} -1 & 2 & -1 \end{bmatrix}$$

Finalmente se tiene la ecuación discretizada:

$$\frac{-U_{i+1} + 2U_i - U_{i-1}}{h^2} = f_i$$

Con $U_0 = U_N = 0$, donde N, representa el número de puntos de la discretización uniforme de Ω, cada uno de tamaño $h = \frac{1}{N-1}$, así produce el sistema:

$$\begin{pmatrix} 2 & -1 & & & & & \\ -1 & 2 & -1 & & & & \\ \vdots & & & \ddots & & & \\ & & & & -1 & 2 & -1 \\ & & & & & -1 & 2 \end{pmatrix} \begin{pmatrix} U_1 \\ U_2 \\ \vdots \\ U_{N-2} \\ U_{N-1} \end{pmatrix} = h^2 \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_{N-2} \\ f_{N-1} \end{pmatrix}$$

En notación matricial $Au = f$. Primeramente veamos como se comportan los métodos iterativos clásicos, en este caso consideremos Gauss-Seidel y a Jacobi, para mayor simplicidad hagamos $f = 0$, asi el sistema $Au = 0$, tiene como solución conocida $u = 0$, por otro lado, sabemos que el error e, dado una solución aproximada v, es de la forma:

$$e = u - v, \tag{3.1}$$

de donde $e = -v$. Tomemos como error inicial para las iteraciónes de Jacobi y Gauss-Seidel, vectores v en los modos de Fourier:

$$v_i = \sin\left(\frac{ik\pi}{N}\right) \tag{3.2}$$

Donde: $0 \leq i \leq N$, y $1 \leq k \leq N-1$, los nodos de Fourier. Así para $k = 1, 3, 6, 12$, se tienen los respectivos errores iniciales, que se muestra en la figura 3–1, donde se puede apreciar que cuando $k = 1$, la frecuencia más baja y con $k = 12$, representa a los errores iniciales con altas frecuencias.

Luego utilizando (**??**), con $i = 100$, es decir, después de 100 iteraciones, tanto para el método de Jacobi como el de Gauss-Seidel, la atenuación de tales errores, se aprecia en las figuras: 3–2 y 3–3.

Observemos primero para el método de Jacobi cómo se comportan estos errores en la figura: 3–2, para un $N = 100$, después de las 100 iteraciones. Las bajas frecuencias, especialmente con $k = 1$, se atenua ligeramente, desde 1.0 hasta 0.9 aproximadamente. Para $k = 3$ disminuye más notoriamente pero aún el valor esta cercano a 0.6. Pero si observamos con $k = 6$ y $k = 12$, que son altas frecuencias se atenuan hasta en un orden de 10^{-3}.

Algo similar sucede cuando se aplica al método de Gauss-Seidel, figura 3–3, aunque se comporta mejor que el método de Jacobi, en el sentido de reducir los errores más rápidamente, tiene un comportamiento similar, atenúa mejor las altas frecuencias y más lentamente las bajas frecuencias.

Otro ejemplo combinado, tanto de altas frecuencias como de bajas, consideremos un error incial de:

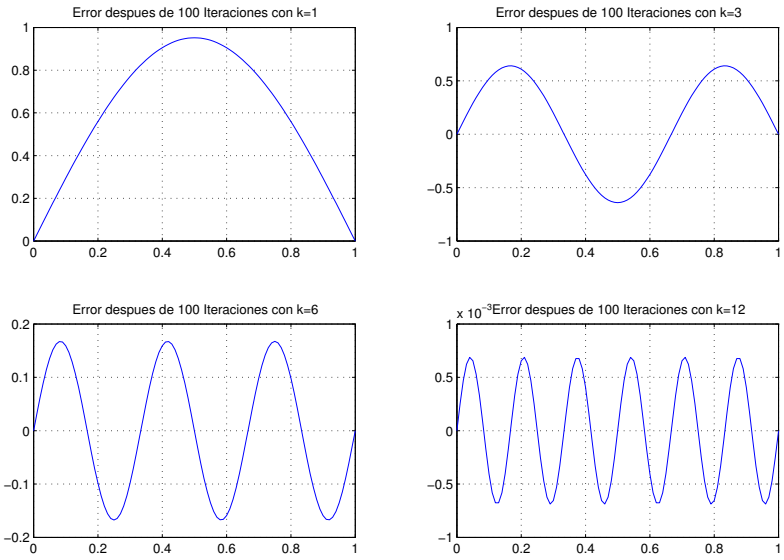

Figura 3–2: Error después de 100 Iteraciones, aplicando Jacobi,para $k = 1, 3, 6, 12$

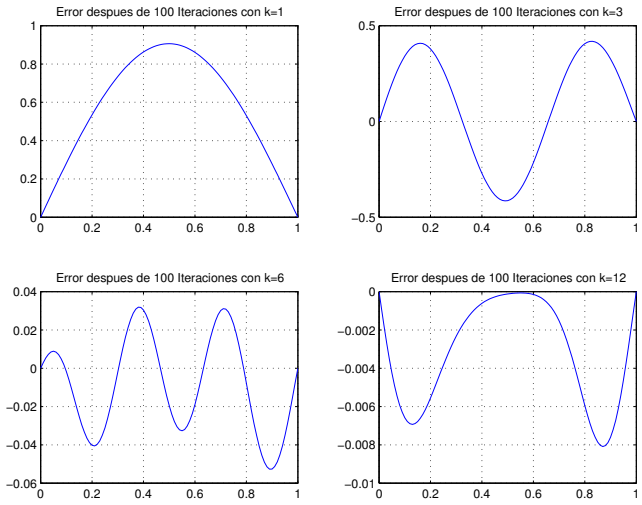

Figura 3–3: Error después de 100 iteraciones, aplicando Gauss-Seidel, para $k = 1, 3, 6, 12$

$$v_i = \sin\left(\frac{2i\pi}{N}\right) + \frac{1}{2}\sin\left((\frac{16i\pi}{N}\right) + \frac{1}{2}\sin\left(\frac{32i\pi}{N}\right)$$

En la figura 3–4, se puede observar claramente que en tan solo 5 iteraciones se atenuan rápidamente los componentes del error que poseen altas frecuencias, mientras que los de bajas frecuencias lo hacen lentamente aún después de 100 iteraciones. Por esta razón este tipo de métodos se les suele llamar *Suavizadores*.

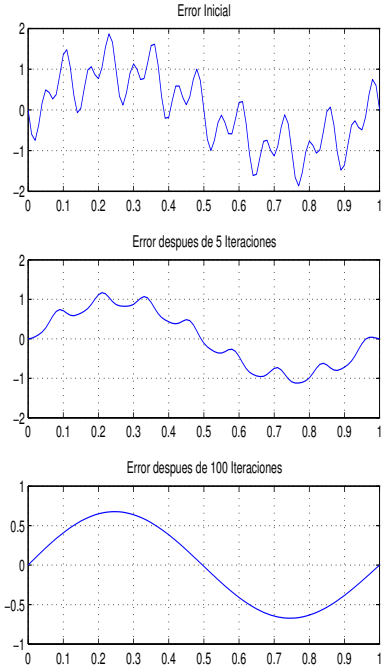

Figura 3–4: Evolución de errores con Gauss-Seidel para 0, 5 y 100 iteraciones

3.3 Multimalla Algebraico

El Multimalla Algebraico comienza su desarrollo notablemente con Brandt, McCormick, Ruge [17, 34, 40, 41]. Puede ser considerado una evolución del multimalla clásico o multimalla geométrico, básicamente en la incorporación de dos elementos importantes, el *Operador de Galerkin* y la Interpolación dependiente del Operador [21, 30]. En el primer caso, como alternativa al proceso natural de seleccionar las mallas gruesas. Por ejemplo si tuviéramos una malla con una resolución de h, lo

usual sería tomar una malla gruesa con resolución $2h$, tal como trabajan los multimalla geométricos, lo cual no resulta apropiado para problemas con coeficientes discontinuos.

Una consecuencia de la incorporación del Operador de Galerkin es que la corrección mediante este operador satisface un principio variacional [34], ampliando así los horizontes en las investigaciones sobre la convergencia del método. Desde el punto de vista práctico este operador además tiene la ventaja de que puede ser construido puramente de modo algebraico, es decir, basado solamente en información de la matriz del sistema.

El otro elemento incorporado es la forma de realizar la interpolación, se plantea que la interpolación sea sobre el esténcil de discretización, lo que permite lidiar con problemas de coeficientes discontinuos, en este tipo de problemas luego de aplicar el suavizador, los errores muestran un comportamiento discontinuo como la solución misma, debido a que la construcción de mallas en el multimalla geométrico transfiere las discontinuidades, mientras uno que tome en cuenta el esténcil se espera que transfiera correctamente estas discontinuidades y asi la combinación de estos dos elementos hace que los multimallas tengan mejor desempeño.

En el multimalla geométrico se pone especial atención al suavizador ya que la construcción o generación de la malla es fijo y/o preestablecida, entonces la tarea recae en la forma de elegir un suavizador apropiado, aquellos errores que no fueron reducidos por el proceso de selección de la malla gruesa, quedaran a responsabilidad del suavizador. Sin embargo en el caso de coeficientes discontinuos o problemas en tres dimensiones, la complejidad del suavizador se incrementará, aquí se pueden considerar suavizar por líneas o planos (suavizar en cada dimensión por separado)

Tabla 3–1: Diferencias entre el multimalla geométrico y el Algebraico

Multimalla Geométrico	Multimalla Algebraico
Las mallas deben estar construidas previamente	No existen mallas, en su lugar son matrices para cada nivel
Dominio sobre mallas estructuradas	Estructuradas y no estructuradas
Tiene problemas en problemas cuyo dominio es de geometria compleja.	Se desempeña bien bien cuando la geometria del dominio es complejo.
Tiene dificultad cuando hay coeficientes discontinuos	Se desenvuelve bien en problemas con coeficientes discontinuos.
La interpolación y restricción es muy sencilla, (ejem. interpolación lineal)	Se debe construir previamente operadores de interpolación y restricción
No tiene fase de configuración	La fase de configuración puede llegar a ser muy costoso.
La selección de malla gruesa es fijo pero se ajusta el suavizador	Se deja fijo el suavizador y se ajusta la selección de malla gruesa

o considerar suavizadores tipo ILU, pero estos pierden muchas de sus propiedades en problemas de tres dimensiones [49]. Por otro lado en el multimalla algebraico se prentende dejar fijo al suavizador y enfocarse en la selección de malla gruesa.

Con estas dos nuevas consideraciones hechas al multimalla geométrico, se puede notar que es posible operar solo con información algebraica, sin depender de la geometría de la malla, es así como dan lugar a los métodos de multimalla algebraico.

Podemos resumir las diferencias entre el multimalla geométrico y el algebraico, en la tabla 3.3

3.4 Componentes del multimalla algebraico

Podemos describir los componentes restringuiendo solo a dos niveles y luego extender a varios niveles recursivamente, estos dos niveles serán:

$$A_h u_h = f_h$$

$$A_H u_H = f_H$$

Consideremos $\Omega^h = \{1, 2, 3, ..., n_h\}$, el conjunto de variables del nivel h, y $\Omega^H = \{1, 2, 3, ..., n_H\}$, las variables en el nivel H, así A_h es una matriz cuadrada de dimensión $n_h \times n_h$, que representa al nivel fino y A_H es una matriz cuadrada de dimensión $n_H \times n_H$, que representa al nivel grueso, u_h y f_h, vectores de dimensión n_h, similarmente para los vectores u_H, f_H de dimensión n_H cada uno. La relación entre Ω^h y Ω^H, se puede deducir de:

$$\Omega^h = C^h \cup F^h \tag{3.3}$$

Donde C^h, representa al subconjunto de variables gruesas y F^h al subconjunto de variables finas en el nivel h. Luego $\Omega^H = C^h$. Para construir A_H se tiene dos posibilidades, o realizar otra discretización de la Ecuación diferencial parcial en la malla H, o utilizar Galerkin:

$$A_H = R_h^H A_h P_H^h \tag{3.4}$$

Donde R y P son operadores denomidados de transferencia.

3.4.1 Operadores de transferencia

Son operadores que transforman vectores de un nivel hacia el otro, R_h^H es el operador de restricción que toma un elemento definido en Ω^h y lo transforma en un elemento de Ω^H, mientras que P opera en sentido inverso, usualmente estan relacionados por $P = R^T$. Para ilustrar la interpolación consideremos el caso de dos

mallas una fina h y una gruesa H.

$$e_i^h = \begin{cases} e_i^H & \text{si } i \in C^h \\ \sum_{k \in p_i} w_{ik}^h e_k^H & \text{si } i \in F^h \end{cases}$$

Donde p_i son las variables interpolatorias, w_{ik}^h, son los pesos asociados a las variables interpolatorias, C^h y F^h son el conjunto de varianles gruesas y finas respectivamente que pertenecen a la malla h, además $p_i \in C^h$, que debe ser pequeño, por lo general las variables gruesas que estan más cercanas a la variable i, para obtener mayor eficiencia. A este tipo de interpolación se le denomina *Interpolacion Directa* [49] porque se realiza con las variables que estan directamente acopladas o relacionadas (en un contexto de teoria de grafos, las variables vecinas).

Por otro lado la *Interpolación Indirecta* es util cuando existen variables finas que no estan acopladas directamente a variables gruesas, esto puede producirse por ejemplo como resultado de una selección de malla gruesa agresiva (que se detalla mas adelante), por lo que será necesario realizar pasos previos, esto es, realizar una interpolación entre las variables finas interconectadas (o buscar otras conexiones adyacentes que si esten acopladas con las gruesas), para luego pasar a las a una interpolacion en la variable deseada, a este proceso se le denomina *interpolación de multipaso*. (vease [49] para mas detalles).

3.4.2 Selección de malla gruesa

Este proceso tiene por objetivo generar los subconjuntos C^h que representa las variables gruesas, estas variables son las que serviran para generar el siguiente nivel o malla más gruesa, sobre estas variables se realizará la interpolación de Ω^H a Ω^h. La selección de estas variables se realiza tomando en referencia variables fuertemente

conectadas. Las variables que dependen fuertemente de otras son considerados finas denotadas por F, y aquellas que influencian fuertemente a otras como gruesas o simplemente C.

Esta consideración porque es en las direcciones donde hay conexiones fuertes donde algebraicamente el error es suave [34, 49, 50] o que varian lentamente.

Para lograr un desempeño eficiente es necesario balancear el número de variables seleccionadas como gruesas, mayor número de variables implica mayor demanda de memoria, se debe considerar limitar el número de variables C.

Para la selección de malla gruesa básicamente podemos mencionar dos enfoques, el primero denominado estandar o clásico que separa las variables de un nivel dado en variables gruesas y finas. El segundo enfoque que consiste en agregar o algomerar variables gruesas para el siguiente nivel.

Selección de malla gruesa estandar

El criterio para diferenciar una variable C de otra F, se realiza a través del concepto de conexiones fuertes, asi una variable i depende fuertemente de j o que j influencia fuertemene a i, si

$$-a_{i,j} \geq \theta \max_{k \neq i} \mid a_{i,k} \mid \tag{3.5}$$

Con $0 < \theta < 1$, originalmente fue definido solo para matrices simétricas del tipo *Matriz M*, esto es una matriz positiva definida y todos los elementos que no estan en la diagonal son negativos, pero puede ser aplicado a matrices más generales. Usualmente $\theta = 0.25$ por consideraciones prácticas, relacionado a la complejidad de

la selección de la malla gruesa, tiene que ver con el tamaño del esténcil.

Este tipo de selección de malla gruesa se realiza tomando en cuenta dos criterios:

1. Para cada variable j que influencia fuertemente a una variable i del tipo F, j es o una variable C o depende fuertemente de una variable k del tipo C, que también influencia fuertemente a la variable i, esto garantiza que una variable F dependa por lo menos de una variable C, también asegura la calidad de la interpolación.
2. No hay dos variables C conectados una a otra, con este criterio se restringe el tamaño del conjunto C.

El seudo código que ilustra esta estrategia se muestra a continuación:

```
SeleccionMallaGruesaEstandar
{
  Seleccionar i del conjunto U
  Insertar i al conjunto C
  Eliminar i del conjunto U
  Mover todos los vecinos de i a F
  Actualizar las medidas de cada i en U
}
```

Donde U representa al conjunto de todas las variables no seleccionadas, inicialmente son todas las variables de Ω, el conjunto C representa las variables elejidas como gruesas, F las variables finas. Algunas implementaciones consideran una cierta medida o peso relacionada con cada variable no seleccionada, para efectos de evitar una selección aleatoria.

Selección de malla gruesa agresivo

Este tipo de construcción de malla gruesa es mas apropiado cuando se tiene pequeños esténciles, porque en estos el requerimiento de memoria es alto. La definición de fuertemente conectados es diferente a la estrategia anterior. Dos variables i, j estan fuertemente conectadas a lo largo de un camino de longitud l, si existe

una sequencia de variables $i_1, i_2, ..., i_l$ con $i_1 = i$, $j = i_l$ y i_k fuertemente conectada a i_{k+1}, para $k = 1, ..., l$. En otras palabras, una variable esta fuertemente conectada si existen al menos p rutas de longitud menor o igual a l. Se puede realizar esta selección de malla gruesa a partir de la primera condición de la estrategia anterior y después calculando la conexiones (p, l), entre las variables C seleccionadas a través de los vecinos F. Si $p = 2$ y $l = 2$ entonces se tiene la selección de malla gruesa conocida como tipo A2, si $p = 1$ y $l = 2$, se tiene tipo A1.

Selección de malla gruesa por aglomeración

También denominado por agregación, donde un agregado es definido por el nodo raiz y sus vecinos. En esta variante el criterio de fuertemente conectado es definido como:

$$| a_{i,j} | > \sqrt{| a_{i,i} a_{j,j} |} \tag{3.6}$$

En esta selección de malla gruesa se selecciona un nodo raiz, que no es adyacente a ningún agregado existente, se repite este proceso hasta que todos los no agregados son añadidos a algún agregado o se consideran como nuevos agregados. Esta estrategia es más eficiente, y menos compleja porque al menos se tiene tres variables en la misma ruta.

3.4.3 Suavizador

Es otro de los componentes importantes del multimalla algebraico, se espera que pueda reducir eficientemente las oscilaciones de los componentes del error, un enfoque estandar solo considera a Gauss-Seidel como suavizador, sin embargo se consideran otros como suavizadores polinomiales, inversas aproximadas, etc.

Dentro de un contexto de multimalla geométrico el término suavizado esta relacionado a la forma en que un determinado error puede ser aproximado a un nivel o malla más gruesa, es decir se considera un error suave si puede ser aproximado a un nivel grueso *predefinido*. Así esta propiedad de suavizado, implica siempre la presencia de dos mallas. Sin embargo en el contexto del multimalla algebraico no existen mallas predefinidas, de modo que esta propiedad de *suavizado*, es adaptada al hecho de como converge un error con respecto a un operador o suavizador S_h, un error será *algebraicamente suave* si converge lentamente, equivalentemente un error es *suave* si ha sido aproximado por medio de una malla o nivel grueso y para ello debe ser construido de forma tal que garantize esta propiedad. Un analisis más detallado se puede encontrar en [29, 49, 53, 58].

3.5 Algoritmo del Multimalla Algebraico

A diferencia del multimalla geométrico, donde la construcción de las diferentes mallas, se realiza de forma separada del mismo método y que es prefijada, en el multimalla algebraico, la generación de los diferentes niveles son construidos como parte del método, pero sin embargo podemos separarlo como dos fases: la de configuración o construcción de las matrices gruesas y operadores de transferencia y la fase de resolución del método. Obviamente el mayor trabajo y tal vez una de las desventajas notables del multimalla algebraico es la fase de configuración.

Fase de configuración

Esta es la fase previa a la fase de resolución, se realiza la construcción de la malla gruesa para cada nivel, para luego construir los operadores de restricción y de interpolación y la malla o matriz. Lo anterior podemos resumir de la siguiente forma:

```
setup(A,u,f,l)
{
```

```
    [C,F] = SeleccionMallaGruesa(A[l-1])
    Construir el operador P[l] Usando  C
    R = Transpuesta( P )
    A[l] =  R[l]*A[l-1]*P[l] )
}
```

Fase de resolución

Para esta fase de existen tres variantes, se tiene el ciclo V como se muestra en el siguiente algoritmo, el ciclo W, y el ciclo F o *Full*.

```
AMG( u, f, Nivel )
{
  Si ( l = l_max)
    Resolver(A[l],u,f)
  caso contrario
  {
    AplicarSuavizador( u, f, n1 )
    d = R(f - A[l]u)
    AMG( v=0, d, Nivel-1 )
    u = u + Pv
  }
  AplicarSuavizador( u,f ,n2 )
}
```

Donde R, y P, son los operadores de restricción e interpolación respectivamente, en este ciclo denominado V, que es el más clásico se empieza desde la malla mas fina, resolviendo recursivamente hasta la malla más gruesa para luego ir ascendiendo a la malla fina, lo que se ilustra en la figura 3–5. Donde $n1$ y $n2$, representan el número de pasos que se aplicara el pre-suavizado y post-suavizado, respectivamente.

3.6 Multimalla como precondicionador

Tanto la experiencia práctica, heuristicamente como algunas investigaciones [35, 44] recomiendan al multimalla como precondicionador, dado que a veces buscar una buena interacción entre el suavizador y la selección de malla gruesa, para obtener un método de resolución independiente, eficiente y con una convergencia

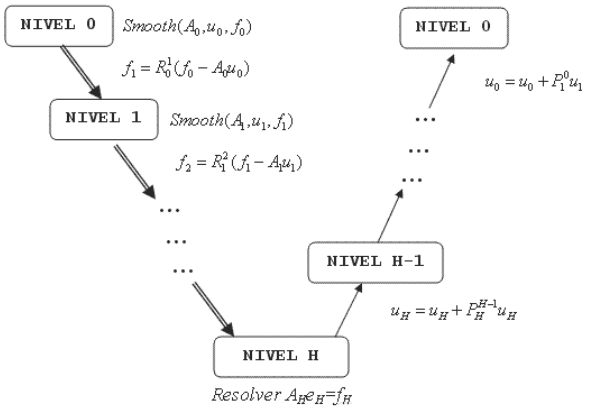

Figura 3–5: Ciclo V de resolución

robusta, no es una tarea fácil. Pero al combinarlo con métodos tales como gradiente conjugado precondicionado, GMRES, etc. se logra mejores resultados, porque de esta forma la fase de configuración será más simple, y segundo porque como ya se vió el multimalla opera en todos los componentes del error.

En el capítulo anterior vimos que se puede obtener un sistema equivalente $\tilde{\mathrm{A}}x = \tilde{\mathrm{b}}$, a partir de $Ax = b$, mediante un precondicionador, de forma tal que el sistema original este expresado como $M^{-1}A = M^{-1}b$, donde M, es el precondicionador, en nuestro caso el multimalla algebraico y la matriz de iteración será $K = I - BA_h$, donde $B = P_H^h A_H^{-1} R_h^H A_H$, con P y R como operadores de interpolación y restricción respectivamente, en un contexto de dos mallas la fina (A_h) y la gruesa (A_H) que puede ser extendida a varios niveles.

CAPITULO 4
MATRICES ESPARCIDAS

La eficiencia de los métodos destinados a resolver sistemas lineales, en este caso provenientes de discretizaciones de ecuaciones diferencias parciales, dependerá en gran medida por el desempeño en las operaciones de algebra lineal, primariamente multiplicaciones de matriz por matriz y matriz por vector, asi el criterio que se de para el manejo y almacenamiento de los datos de matrices influirá en el rendimiento del algoritmo.

En el caso de matrices vinculadas a discretizaciones de EDP, estas contienen gran cantidad de ceros, que no deben ser manipulados ni almacenados, este tipo de matrices donde muchas de las entradas son iguales a cero se les denomina como Matrices Esparcidas. El formato apropiado para almacenar y manipular datos de una matriz depende de cada aplicación.

4.1 Formatos de almacenamiento de Matrices esparcidas

Existen los que consideran almacenamiento por elemento, bloques, filas, columnas, coordenadas, diagonales, etc.

4.1.1 Matriz esparcida en formato de coordenadas

No tiene ninguna sofisticación, es la más simple se necesitan 3 vectores de la misma dimensión

1. I: para almacenar los índices de las filas.

2. J: para almacenar los índices de las columnas.
3. V: el valor de la entrada en la matriz.

Si se tiene la siguiente matriz:

$$\begin{pmatrix} 0.95 & -51.034 & 8 \\ 0 & 0.67 & -4.69 \\ 0.3 & 12 & 0 \\ 0 & 0 & 21 \end{pmatrix}$$

En formato de coordenadas se tendrá los siguientes vectores:

I	1	1	1	2	2	3	3	4

J	1	2	3	2	3	1	2	3

V	0.95	-51.034	8	0.67	-4.69	0.3	12	21

4.1.2 Matriz esparcida en formato de filas comprimidas

Conocido como CSR (del ingles Compressed Sparse Row), es el formato más usado de almacenamiento, no consideran ningún tipo de información sobre la esparcidad de la matriz, solo almacena los valores que son distintos de cero en forma contigua, por lo cual es necesario hacer un direccionamiento indirecto para ubicar un elemento particular, o para efectuar las operaciones, siendo así, necesitan almacenar información extra, que contribuya a la ubicación de un elemento de la matriz. En total se tendrá tres vectores para almacenar en este formato:

1. Un vector para almacenar las entradas o valores de la matriz.
2. Un vector para almacenar los índices de las columnas de cada entrada.
3. Un vector para almacenar el apuntador de fila.

Si tomamos la misma matriz del formato anterior, en CSR se representaria mediante los siguientes vectores:

Val:	0.95	-51.034	8	0.67	-4.69	0.3	12	21

Col:	1	2	3	2	3	1	2	3

Af:	1	4	6	8	9

Donde Val es el vector de valores, diferentes de cero de la matriz construida a través de un recorrido por filas y de izquierda a derecha de la matriz original. Col, son los índices de las columnas que corresponden a los elementos del anterior vector, ejemplo para la entrada 3 en el vector Val, le corresponde el índice de columna ubicado también en la entrada 3 dentro del vector Col. Finalmente Af, representa el apuntador al índice del vector Val donde empieza una nueva fila en la matriz original.

Observemos los dos primeros elementos del vector de apuntadores de filas, en este ejemplo son 1 y 4, restando $4 - 1$, obtenemos el número de entradas en la primera fila, que son diferentes de cero, si observamos los 3 primeros valores en el vector Col, estas corresponden precisamente a las entradas: $(1, 1)$, $(1, 2)$ y $(1, 3)$, además 4 representa el índice dentro de Col y Val, donde empieza la segunda fila, a continuación seguimos recuperando el siguiente valor, que es 6, restamos 6 menos la entrada anterior que es 4, obtenemos 2 que representa el número de entradas en la segunda fila de la matriz original y también 6 representa el inicio de la siguiente fila es decir la tercera y asi sucesivamente, podemos reconstruir la matriz original.

Lo anterior es considerando que la matriz no es simétrica, si la matriz fuera simentrica solo se almacenaria la triangular superior o la inferior.

Si se almacenara la matriz considerando los ceros se tendría un costo del orden de $O(n^2)$, mientras que con este formato solo se tiene: $2nz + n + 1$, con n la dimension de la matriz y nz el número de entradas no ceros de la matriz.

4.1.3 Matriz esparcida en formato de columnas comprimidas

También conocida como matriz esparcida en formato Harwell-Boeing o simplemente CSC (del ingles Compressed Sparse Column), algo similar al anterior formato a diferencia que en este caso se almacenan los índices de las filas y el apuntador ahora es el de columnas, por tanto también usa 3 vectores para manejar la información de la matriz. Así la matriz del ejemplo anterior, en formato CSC sería:

Val:	0.95	-51.034	8	0.67	-4.69	0.3	12	21

Fil:	1	3	1	2	3	1	2	4

Ac:	1	3	6

4.1.4 Matriz esparcida en formato de diagonales

Si en la diagonal y/o en algunas subdiagonales se tiene entradas diferentes de cero, donde el número de entradas por fila es casi constante, este formato es el más adecuado. Las diagonales se almacenan una a continuación de otra comenzando con la diagonal que está más inferior hasta la más superior, una observación importante es que las entradas en la diagonal pueden contener ceros.

Se elimina la utilización de los vectores para identificar las filas y las columnas, porque solo se almacenarán las diagonales de la matriz, lo que puede contribuir en la eficiencia de la multiplicación matriz por vector.

Este tipo de matrices es más apropiada para super-computadoras vectoriales, y se utilizan en mallas construidas de discretizaciones provenientes de diferencias finitas y elementos finitos, construidos en base a un producto tensorial.

Ejemplo si se tiene la siguiente matriz:

$$\begin{pmatrix} 3 & -5 & 0 & 0 & 0 \\ 2 & 10 & -4 & 0 & 0 \\ 0 & 1 & 5 & 4 & 0 \\ 0 & 0 & 2 & 2 & -8 \\ 0 & 0 & 0 & 1 & 4 \end{pmatrix}$$

En formato de diagonales, la anterior matriz queda representada como:

val(:,-1)	2	1	2	1	0

val(:,0)	3	10	5	2	4

val(:,1)	0	-5	4	0	-8

Donde el primer vector $val(:,-1)$, representa la diagonal que se encuentra en la parta inferior de la diagonal principal de la matriz, $val(:,0)$ la diagonal principal y $val(:,1)$ la diagonal por encima de la principal.

4.1.5 Filas comprimidas por bloques

Este formato es apropiado cuando se tiene un patron esparcido por bloques cuadrados en las entradas de la matriz, estos bloques tipicamente vienen de discretizaciones que con varios grados de libertad, puede implementarse modificando los formatos anteriores, en este caso serían BCRS y BCCC, etc.

4.2 Grafo asociado a una matriz

Los grafos asociados a las matrices esparcidas contribuyen en gran medida a las técnicas asociadas con matrices esparcidas, porque existe una teoría bastante desarrollada y consistente sobre grafos, lo cual resulta ser un ingrediente clave en estrategias tales como paralelización, permutaciones y reordenamientos.

La forma de representar una matriz en términos de los elementos de un grafo sería, los vértices del grafo representan a cada variable de la malla, equivalentemente, a cada variable en la matriz (fila o columna), la relación que existan entre el vértice i y el vértice j, está determinada por existencia de un valor distinto de cero en la entrada (i, j) de la matriz, que representa la relación de la variable i con la variable j, o mejor aún una incógnita j en la ecuación i.

Muchos de los algortimos que se utiliza en los multimalla algebraicos se basan fuertemente en el grafo asociado a la matriz esparcida, ejemplo la rapidez en la cual sean procesados operaciones de multiplicación matriz por vector y matriz por matriz dependerán de como esten implementados los algoritmos para el grafo asociado a la matriz respectiva.

En un ámbito de computación en paralelo, al contar con la información del grafo, se puede descomponer o particionar a través de sofisticados algoritmos de forma que el balance de carga entre los procesadores sea lo más uniforme posible, brindando mayor eficiencia en el cómputo y la utilización de los procesadores.

Mayor información sobre matrices esparcidas, su almacenamiento y demas tópicos relacionados se pueden encontrar en [13, 46].

CAPITULO 5
PROGRAMACIÓN GENÉRICA

Programación genérica es la programación con conceptos [43], donde un concepto es una familia de abstracciones relacionados por un conjunto de requerimientos en común. Este paradigma está basado en la generalización o abstracción de algoritmos y estructuras de datos [10, 12], mediante el cual se puede escribir algoritmos donde los tipos de datos tendrán el comportamiento de un parámetro [11]. El término abstracción es fundamental para este paradigma, los algoritmos que se desarrollen serán en términos generales o conceptos y no limitado a un tipo datos específico.

La programación genérica también se puede encontrar con el nombre de programación politípica [33], ya que es la implementación de algoritmos con funcionalidades semejantes pero aplicado a diferentes tipos de datos, se puede encontrar más información bajo los nombres de polimorfismo estructural, programación paramétrica de tipos, polimorfismo polinomial, etc.

5.1 Utilización de la Programación Genérica

Desarrollar software vía la programación genérica se puede caracterizar en dos fases [12], se empieza en la abstracción buscando patrones en común removiendo los detalles poco relevantes para quedarse con lo esencial o genérico, esta etapa se le denomina generalización como resultado se tendrá un conjunto de reglas o métodos, que podrán ser aplicado en la segunda fase de especialización para instanciar a una

estructura de datos especifico.

5.2 Implementación de la Programación Genérica

Muchos de los lenguajes modernos traen soporte para la programación genérica, como por ejemplo: C++, ADA, ML, Haskell, Eiffel, Java, C♯. Un ejemplo de ellos es la implementación de las librerias estándar de C++, conocidas por STL [8, 42], por sus siglas en ingles. Los *templates* o plantillas en C++ fueron diseñadas para soportar la programación genérica, en tiempo de compilación, se han comprobado que estos mecanismos, son muy útiles en la generación de código para aplicaciones en científicas [36, 39], a demas que el C++ por si mismo ya tiene gran aceptación por la comunidad de Computación Científica.

En C++ un ejemplo de implementación de programación genérica son la libreria STL("Standar Template Library"), ésta fué desarrollada en base a dos elementos del lenguaje C++, las Clases y las Plantillas. Los componentes de esta libreria se pueden clasificar por: Contenedores, Algoritmos e Iteradores.

Un Contenedor o plantilla son clases genéricas, para contener a objetos de varios tipos, ejemplo el contenedor Vector, que puede soportar tipos como enteros, reales, etc. Los algoritmos son los operadores que actuaran sobre estos contenedores, no pertenece a ningun tipo en particular. Los iteradores es el componente nexo entre los dos anteriores, para recorrido o extracción de elementos.

CAPITULO 6
ABSTRACCIÓN MATEMÁTICA

En este capítulo mostraremos la equivalencia entre los elementos o partes del multimalla algebraico en términos de implementación, con las consideraciones de la programación orientada a objetos y utilización de la programacón genérica que provee C++, en otras palabras *La Representación Computacional* de su homólogo matemático, mostrado en el capítulo 3, que da como resultado a AMGLib++.

Una analogía se muestra en la figura 1–1 del capítulo 1, como un conjunto de piezas de un rompecabezas, donde cada pieza debe ser ensamblada de forma tal que juntas formen la figura del rompecabezas, uno podría reemplazar una pieza o varias piezas y de esta manera variar la imagen del rompecabezas, pero debe cuidar que esta luego se acople perfectamente con el resto de las piezas. En el caso de AMGLib++, cada pieza del rompecabezas representa a un componente del multigrid algebraico, existiendo una pieza central denomida núcleo que representa el algoritmo de multimalla algebraico.

En la figura 6–1, se muestra las partes que conforman a este entorno, en principio se puede identificar, en la parte derecha, el núcleo de la librería y la parte izquierda de la misma figura, delimitada por las lineas entrecortadas, los módulos que en este caso son módulos o clases paramétricas, denominadas así, porque pueden ser reemplazadas y/o personalizadas, siguiendo con la filosofía de la librería.

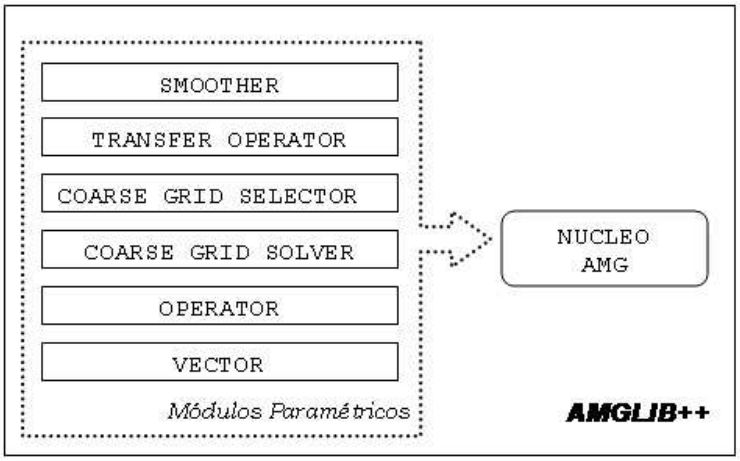

Figura 6–1: AMGLib++: Sus partes, el núcleo y los modulos paramétricos.

6.1 Ámbito AMG o Núcleo del AMGLib++

El núcleo de la librería es la clase AMG, que implementa el algoritmo de configuración y el algoritmo de resolución, aqui se realiza toda la interacción entre los diferentes componentes del multimalla algebraico. AMG, está provista de métodos de clase (se entiende por métodos de clase a las funciones que forman parte de la implementación computacional o código de programa de la clase) que permiten establecer las instancias de las clases con las cuales trabajará, ejemplo si se tiene una clase particular de suavizador, que llevará a cabo el proceso de pre-suavizado (vease el capítulo 3 para detalles sobre Pre-Suavizador y Post-Suavizador) se debe establecer el vínculo con la función apropiada, en este caso *set_pre_smoother(mySMOOOTHER)*, donde mySMOOTHER representa una implementación hecha por el usuario. Además de ello, AMG contiene métodos de clase que interactuan con el resto de componentes, ejemplo, para el proceso de *Selección de malla gruesa*, se invocará: *myCoarsening->run*, donde myCoarsening es una implementación de alguna estrategia hecha por el usuario o las pre construidas, que vienen con la librería y *run*, el método de clase que ejecuta la estrategia de selección de malla gruesa. En la sección de *Conexión de componentes en AMGLib++*, se verá los pormenores de

las relaciones entre componentes de la librería.

6.1.1 La Clase AMG

la implementación del nucleo esta representada por la clase AMG, cuya definición se muestra a continuación:

```
1  template <typename VECTOR,
2           typename OPERATOR,
3           typename COARSE_GRID_SELECTOR,
4           typename TRANSFER_OPERATOR,
5           typename SMOOTHER,
6           typename COARSE_GRID_SOLVER>
7  class AMG: public Solver<OPERATOR,VECTOR>
8  {
9  public:
10   AMG();
11   virtual ~AMG();
12
13 void set_presmoother( SMOOTHER* pre );
14 void set_postmoother( SMOOTHER* post );
15 void set_min_size( int mins );
16 void set_max_level( int maxl );
17 void set_coarse_grid_selector( COARSE_GRID_SELECTOR* cgs );
18 void set_transfer_operator( TRANSFER_OPERATOR* T );
19 void set_num_pre_smoothing( int nps );
20 void set_num_post_smoothing( int npps );
21 void set_coarse_grid_solver( COARSE_GRID_SOLVER* cgs );
22 virtual void setup();
23 virtual void solve();
24 virtual void smooth( OPERATOR* A,VECTOR* u,
25                      VECTOR* f,int type,int level );
26 virtual void mgm( VECTOR* u,VECTOR* f,int level );
27 virtual void solve_coarse( OPERATOR* A,VECTOR* u,VECTOR* f );

28 protected:
29   SMOOTHER*           pre_smoother;
30   SMOOTHER*           post_smoother;
31   vector<OPERATOR*>   R; /**< restriction operators */
32   vector<OPERATOR*>   P; /**< prolongation operators */
33   vector<OPERATOR*>   A_H; /**< coarse grid levels */
34   COARSE_GRID_SELECTOR* coarse_grid_selector;
```

```
35   TRANSFER_OPERATOR*    transfer_operator;
36   COARSE_GRID_SOLVER*  coarse_solver;
37
38 private:
39   int min_size;
40   int max_level;
41   int num_presmoothing;
42   int num_postsmoothing;
43 };
```

En las lineas 1 hasta la 6 se encuentra la definición usual en C++ para la implementación de la programación genérica, en este caso una plantilla genérica para la clase AMG, especificada por la palabra reservada de C++ *TEMPLATE*. Otra palabra reservada de c++ *typename* define a un tipo abstracto o paramétrico que el usuario puede reemplazar.

En la linea 7 esta la definición de la clase y además se muestra que es una clase derivada de otra, denominada *SOLVER* (Vease Herencia de clases más adelante), el cual a su vez es otra plantilla, que utilizará tanto el tipo abstracto OPERATOR y VECTOR que se indique a la clase AMG.

En las lineas desde la 13 hasta la 21, estan las funciones para establecer valores a los atributos de la clase AMG correspondientes, como sus nombres lo indican, en las lineas siguientes hasta la linea 26, se muestran las funciones principales de la clase (los que representan a su similar en el modelo matemático).

Desde la linea 29 hasta la 42 se encuentra la definición de los atributos de la clase, que son la representación computacional de los componentes del método multimalla algebraico.

6.2 Módulos paramétricos

Conservando siempre una relación con su similiar matemático, no es muy difícil, deducir el resto de componentes computacionales del MMA, y las funciones que estas cumplen, sin embargo brevemente los mencionaremos, como se muestran en la figura 6–1. Los nombres adoptados estan en ingles.

1. VECTOR, como su nombre lo indica representa a un vector pero en su sentido amplio, puede ser vectores de reales, vectores de complejos, o un tipo de datos que se especifique.
2. OPERATOR, en este caso es el tipo abstracto sobre el cual todos trabajan, haciendo similitud con su homologo matemático seria aquella entidad que representa a un operador de discretización que para el caso del MMA esta representada como una matriz.
3. SMOOTHER, Cuya principal función es realizar unas cuantas iteraciones de algún método de resolución de sistemas lineales de ecuaciones, como se establecio en el capítulo 3.
4. TRANSFER OPERATOR, Es el encargado de realizar la construcción de los operadores de transferencia, para realizar la restricción y la interpolación.
5. COARSE GRID SELECTOR, Este tipo abstracto representa al proceso de selección de malla gruesa.
6. COARSE GRID SOLVER, Por lo general se elije un método directo tal como eliminación gaussiana para la resolución en el nivel más grueso, pero sin embargo se deja a libertad.

6.3 Conexión de componentes en AMGLib++

Dado que se trabajará con politipos, es preciso establecer, los protocolos de comunicación entre estas clases, esto se puede interpretar como requisitos mínimos si se desea utilizar AMGLib++.

Hasta ahora no se conoce la forma como realizará su trabajo cada tipo abstracto, tal como establece la programación genérica, a excepción de la clase AMG, lo importante aquí es que las instancias que actuen como componentes del MMA, deben cumplir con los siguientes requerimientos:

6.3.1 Relación AMG con OPERATOR

Operator debe proveer las siguientes funciones:

1. *GET_NUM_NODES*: Que permita saber cuantas variables componen el operador, cuando la matriz es cuadrada, que ocurre en los operadores que representan a los diferentes niveles gruesos (A_0, A_1, ..., A_H), (vease fig. 3–5 en el capitulo 3. En caso que la matriz no sea cuadrada, se debe considerar: *GET_NUM_ROWS* y *GET_NUM_COLS*, para obtener el numero de filas y columnas respectivamente, esto ocurre con los operadores de interpolación y restricción, donde el número de filas excederá al de columas, y viceversa para la restricción.
2. *MUL*, que permita realizar la multiplicación de 3 matrices, especificamente $A_H = R_h^H A_h P_H^h$, lo cual es necesario cuando se desea construir un nivel grueso a partir de uno fino, en un modelo de 2 niveles, utilizando el principio de Galerkin (Vease capitulo 3, para mayor información), el resultado de este método de clase debe generar un cuarto operador, el cual representa al nivel H o grueso.
3. *MAT_MUL*, que permita realizar el producto de matriz por vector, al estilo LAPACK, que se usa por ejemplo cuando se restringe o se interpola de un nivel fino a grueso y viceversa respectivamente.
4. *TRANSPOSE*, adicionalmente si se desea almacenar el operador de Restricción, que por lo general es construido mediante la transpuesta del Interpolador.

6.3.2 Relación AMG con COARSE GRID SELECTOR

1. *SET_OPERATOR*, para poder establecer sobre que Operador, o matriz, se realizará la selección de malla gruesa.
2. *RUN*, Para ejecutar el procedimiento de selección de nodos gruesos y nodos finos, del nivel establecido en el requerimiento anterior.

6.3.3 Relación AMG con TRANSFER OPERATOR

1. *SET_COARSENING*, para establecer que clase u objeto, es el que tiene las variables C y F, seleccionadas, para construir los operadores de transferencia.
2. *INTERPOLATOR*, una función que devuelva el operador de interpolación en base al item anterior. Se prevee que también pueda especificarse otra forma además de la forma usual $R = P^T$, de construcción de la restricción.

6.3.4 Relación AMG con SMOOTHER

SMOOTHER es la implementación de un método de resolución, se debe especificar la información necesaria para resolver el sistema lineal $Ax = b$, como son A la matriz del sistema, x y b, los vectores solución y de lado derecho, respectivamente. Estos métos deben ser:

1. *SET_MATRIX*, para establecer sobre que matriz se trabajará.
2. *SET_SOLUTION_VECTOR*, para especificar el vector solución que se devolverá.
3. *SET_RIGHT_HAND_SIDE*, para precisar el vector de lado derecho del sistema.
4. *SOLVE*, que ejecuta el método de resolución del sistema.

6.3.5 Relación AMG con COARSE GRID SOLVER

Similar al anterior, solo que en este caso no esta restringido a métodos iterativos. Usualmente se utiliza la eliminación Gaussiana para resolver el sistema lineal. Se deben proveer métodos similares al de la clase SMOOTHER.

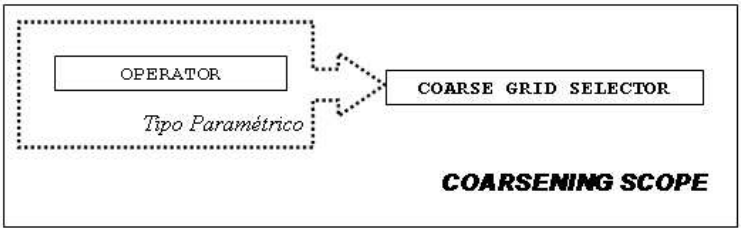

Figura 6–2: Ámbito de selección de malla gruesa

6.4 Ámbito de COARSENING

Otras generalizaciones que se contemplan son sobre la selección de malla gruesa, definimos COARSENING como la clase que representa o implementa la estrategia de selección de malla gruesa, a su vez es una clase abstracta porque no implementará el proceso como tal sino que servirá para que se puedan heredar otras clases que si implementen el proceso, en la sección de Jerarquía y herencia se detallará más acerca de la relación de clases.

En esta sección usaremos algunas convenciones de la representación de matrices u operadores basados en la teoria de grafos, vease [46] para más información al respecto. En la fig. 6–2, se representa gráficamente. En la parte derecha, esta la clase paramétrica *OPERATOR*, no es necesario conocer como es la clase pero si los requerimientos o funciones que debe contener, que son:

1. *GET_NUM_NODES*, el mismo especificado en la relacion de *AMG* con *OPERATOR*.
2. *GET_NUM_NEIGHBORS*, dado un nodo v, para devolver el número de nodos que tiene como vecinos un nodo dado.
3. *INIT_EXTRACTION_NEIGHBORS*, para inicializar el proceso de extracción de variables o nodos (podemos referirnos también como nodo, según la representación

de grafos, para los operadores o matrices) de interpolación de un nodo dado, además debe retornar el número de nodos vecinos.

4. *NEXT_NEIGHBOR*, para extraer el siguiente nodo vecino a uno especificado en el item anterior, esta función devolverá cada uno de estos nodos vecinos, aquí se puede especificar algún criterio para diferenciar de aquellos nodos que tienen conexiones fuertes tal como se especifica en [49].

En este ámbito la implementación de un nuevo algoritmo de selección de malla gruesa no necesariamente tiene que cumplir con las especificaciones dadas aqui, ya que uno podría crear un algoritmo distinto y reemplazarlo o especificar el mismo directamente en la clase AMG, recordemos que *COARSE_GRID_SELECTOR*, es un tipo paramétrico.

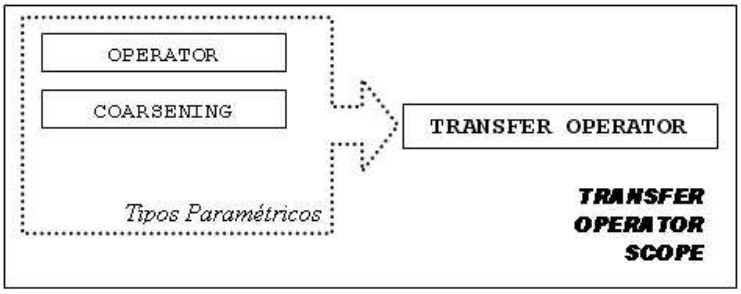

Figura 6–3: Ámbito de selección de generación de operadores de transferencia

6.5 Ámbito TRANSFER OPERATOR

Éste ambito sirve para la implementación del proceso de generación de operadores de transferencia, se debe dar una relación con el ámbito de COARSENING, porque de ahi se extraerá los conjuntos de nodos gruesos y finos, los métodos que se deben contemplar son:

1. INIT_GET_COARSE, en caso de que se necesite realizar una inicializacion previa a la extraccíon de variables C, aparte de ello que permitan saber el número de variables del tipo C.
2. INIT_GET_FINE, similar al anterior pero para variables del tipo F.
3. GET_NEXT_COARSE, extraer el siguiente nodo C.
4. GET_NEXT_FINE, extraer el siguiente nodo F.
5. GET_INTERPOLATION_NODES, obtiene el número de variables de interpolación, para un nodo dado.
6. NEXT_INTERPOLATION_NODE, para obtener el siguiente nodo disponible de interpolación de un nodo dado.

Básicamente, lo que se prentende es obtener una forma de extraer las variables C y F, generadas en el ámbito de *COARSENING*, así como también de obtener los nodos que serviran para interpolar, es decir dado un nodo $v_i \in F$, se devolverá $C_i = N_i \cup C$, donde N_i, son los vecinos del nodo v_i.

Nuevamente recalcar que esto no es rígido, uno podría hacer lo mismo que para el ámbito anterior, que sería crear completamente una nueva clase encargada de generar los operadores de transferencia.

Una vez obtenido el peso que corresponde a cada entrada del operador de interpolación, que son los pesos $w_{i,j}$, tal que i representa el i-ésimo nodo fino, y j el j-ésimo nodo C_j. Finalmente el peso calculado para las entradas en la matriz de interpolación se debe almacenar en un nuevo *OPERATOR*, para lo cual es necesario, tener acceso al elemento (i, j), que podria ser mediante $OPERATOR->SET\_VALUE$, que asigna un valor dado en una posición (i, j) dados.

En C++ se hace uso de la implementación de operadores sobrecargados vease [22, 38].

Este ámbito queda representado gráficamente en la figura 6–3.

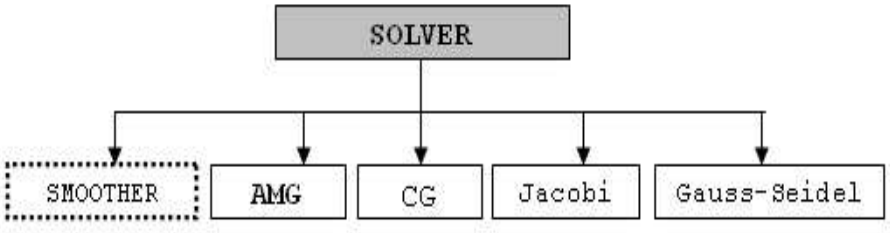

Figura 6–4: Relación entre los componentes del AMG en la fase de resolución

6.6 Herencia de clases

Otro de los beneficios que aporta la programación orientada a objetos es sobre la reutilización de parte o todo el código de una clase, así en AMGLib++, se establece una herencia de clases. Subrayamos que uno de los objetivos de la librería es la implementación de un *método de resolución*, de sistemas lineales, en este caso grandes sistemas esparcidos, pero también Jacobi, Gauss-Seidel, Gradiente Conjugado, lo son, por tanto crear una jerarquia de clases en este punto es adecuado, como se muestra en la figura 6–4, donde se puede observar que en la parte derecha se encuentra las clases Jacobi y Gauss-Seidel, que son métodos de resolución por tanto descienden de una clase base abstracta, *Solver*, lo mismo para el Gradiente Conjugado (CG) y también la clase de multimalla algebraico (AMG), y en general, incluso podria servir como plantilla para futuros suavizadores (que se representa por las lineas punteadas, en el lado derecho de la misma imagen, debido a que es politípico).

Por tanto se pre-establece los siguientes funciones propias de una clase encargada de realizar la solución del sistema $Ax = b$, orientadas a un comportamiento de un método iterativo en general:

1. set_matrix
2. set_initial_guess
3. set_right_hand_side
4. set_tolerance
5. set_max_iteration
6. set_solution_vector

Los que sirven para inicializar con los elementos necesarios para que solucione el sistema lineal. Por tanto en AMGLib++ tanto los suavizadores, el gradiente conjugado precondicionado, así como el mismo AMG, poseen estas funciones, necesarias para la inicialización de la clase.

6.7 Instanciación de AMG

Para realizar una instanciacíon del AMG, es necesario que previamente esten creadas las clases paramétricas que conforman las partes del MMA, anteriormente descritas, para luego proceder con la clase AMG como se aprecia en el siguiente código:

```
1   // Crear el operador para la malla fina y cargar datos
2   CSR_Matrix* A = new CSR_Matrix;
3   A->load_from_csr_file( "laplacian1.txt" );

4   // Definir una estrategia de seleccion de malla gruesa
5   RS* C = new RS_Coarsening;

6  // Definir Gauss-Seidel como suavizador
7  typedef Gauss_Seidel<CSR_Matrix,double> GS;
8  GS* gs = new GS;
```

```
9  // Definir la clase para los operadores de transferencia
10 typedef Transfer_Operator<RG_Coarsening,CSR_Matrix> Transfer;
11 transfer* T = new transfer;

12 // Definir la clase encargada de resolver la malla mas gruesa
13 typedef Gaussian_Elimination<CSR_Matrix> Gaussian_Elimination;
14 Gaussian_Elimination* ge = new Gaussian_Elimination;
```

En estas lineas se definieron los tipos y las variables para cada tipo, que servirán para la clase genérica AMG, la cual quedara definido como:

```
15 AMG<double,
16     CSR_Matrix,
17     RS_Coarsening,
18     Transfer,
19     Gauss_Seidel,
20     Gaussian_Elimination> amg;
```

Esto representa a la instanciación de un multimalla algebraico para un conjunto de tipos de datos dados por el usuario, que usará, reales (doubles) como elementos de un vector, un operador implementado en formato de matriz esparcida del tipo filas comprimidas, el coarsening que se usará esta implementado en la clase RS_Coarsening, el clásico Ruge-Stüben, el Suavizador a utilizarse es Gauss-Seidel y finalmente el método de solución para la malla gruesa es la eliminación Gaussiana implementada en la clase Gaussian_Elimination.

A continuación se debe establecer los parametros necesarios como son tamaño mínimo para la malla gruesa, el número de pasos para el pre y post suavizado, etc.

```
19 // Configurar los paremetros necesarios para el MMA
20 amg->set_min_size( 500 );            // Malla mas gruesa
21 amg->set_coarse_grid_selector( C );  // Seleccion de malla gruesa
22 amg->set_transfer_operator( T );     // operadores de transferencia
23 amg->set_coarse_grid_solver( ge );   // Para la malla mas gruesa
24 amg->set_matrix( A );                // Matriz de la EDP
25 amg->set_presmoother( gs );          // Pre suavizador
```

```
26 amg->set_postmoother( gs );          // Post Suavizador
27 amg->set_num_pre_smoothing( 2 );   // Pre suavizado
28 amg->set_num_post_smoothing( 2 );  // Post suavizado

29 // Fase de configuracion
30 amg->setup();
```

En el caso de utilizar al multimalla algebraico como un precondicionador, que es lo recomendable, para métodos como el gradiente conjugado precondicionado, el cual se muestra a continuación:

```
31 // Definir una instancia para el Gradiente Conj. Precond.
32 PCG<Operator_CSR,double,am> pcg;
33 pcg.set_matrix( A );
34 pcg.set_preconditioner( amg );

35 // Configurar parametros para la fase de resolucion
36 pcg.set_initial_guess( u0 );
37 pcg.set_right_hand_side( f );
38 pcg.set_tolerance( 1e-6 );
39 pcg.set_max_iteration( 20000 );
40 pcg.set_solution_vector( u );

41 // Resolver!!!
42 pcg.solve();
```

6.8 Clases preconstruidas en AMGLib++

En la librería se implementan algunas clases específicas utilizando la *Librería de Estandar de Plantillas del C++*, conocida ampliamente como *STL* (por sus siglas en ingles Standard Template Library), que se detallan a continuación:

6.8.1 Operator_CSR

Es una implementación de matrices esparcidas, tomando una representación de listas para las filas de la matriz, esta contiene una lista de *Nodos*, que representan las variables o incógnitas del sistema lineal esparcido, cada una de las cuales conserva otra lista de *Aristas*, los vecinos a cada uno de los vertices.

Las operaciones implementadas son las establecidas en la sección anterior, donde la operación de multiplicación de matrices se realiza de la forma como se observa en la figura 6–5.

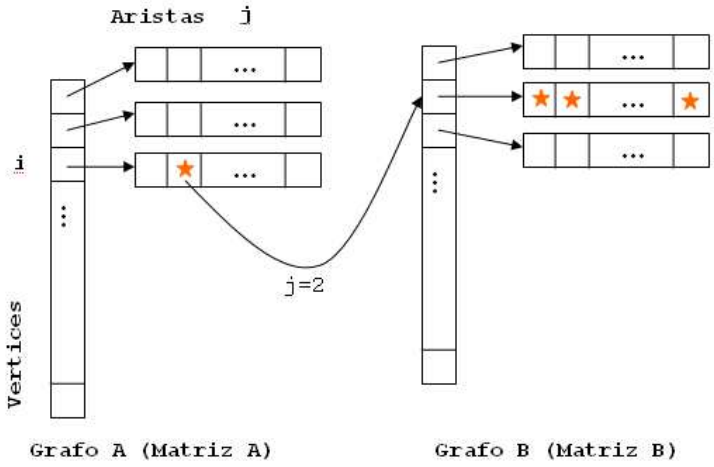

Figura 6–5: Multiplicacion de matrices usando la representación de grafo.

De la figura 6–5, Se multiplica cada nodo $w_j \in v_i$ en el Grafo A (donde v_i, es la i-ésima arista de A), con cada nodo $k \in Aristas(v_j)$ del Grafo B, el resultado se acumula en la matriz resultante, esto en el caso de la multiplicación de dos matrices, del mismo modo, no resulta difícil extender esta idea a la multiplicación de tres matrices.

Si consideramos que los Grafos G_A, G_B, G_C, que representan a las matrices cuadradas A, B y C, de tamaño n, con un patron de esparcidad uniforme igual a s, el costo de multiplicar estas matrices en esta forma sería del orden de $O(s^3 n)$.

6.8.2 La clase rnd_coarsening

Implementa una estrategia de coarsening aleatoria, orientada exclusivamente para la clase Operator_CSR, detallada lineas arriba, la forma de eligir los variables gruesas se realiza aleatoriamente, esto en teoria no resulta ser una buena técnica de selección de malla gruesa, sin embargo sirve para efectos de demostración de la

implementación de *Clases Definidas por el Usuario.*

6.8.3 La clase simple_coarsening

Esta también es otra técnica de selección de malla gruesa, que toma el primer nodo que este disponible en el conjunto de variables que no fueron selecciónadas aún, esta técnica tiene mejores resultados que la anterior como se puede apreciar en el capítulo de experimentos. Esta clase también trabaja directamente sobre la clase *Operator_CSR* de AMGLib++.

6.8.4 La clase RS_coarsening

También trabaja sobre la clase *Operator_CSR* que implementa la estrategia descrita por J.W.Ruge and K.Stüben [50].

CAPITULO 7
EXPERIMENTOS

En esta sección se mostrará los resultados obtenidos en la solución de algunos problemas típicos para el método de multimalla algebraico, que puedan mostrar el comportamiento y la utilización de la librería.

Para los experimentos realizados en este capítulo se adopta una combinación del método de gradiente conjugado precondicionado con el multimalla algebraico, por lo tanto se realizaron las adaptaciones necesarias para tal fin. La tolerancia seleccionada como criterio de parada del gradiente conjugado fue de 10^{-7}, como error relativo. Por otro lado el tamaño mínimo de la malla más gruesa fue de 500 incógnitas.

7.1 Experimento 1

En este experimento mostraremos el problema más típico, el cual fue abordado por los multimallas en sus inicios, cabe señalar que un multimalla geométrico sería mas apropiado para este problema, pero sin embargo la idea en este experimento es mostrar la utilización y combinación de los diferentes elementos y como realizar la personalización de los mismos, en el contexto del AMGLib++.

Ecuación:

$$-\Delta u = f \tag{7.1}$$

Con $u \in \Omega = (1, N) \times (1, N)$ y condiciones de borde: $u(x) = 0$, con $x \in \partial\Omega$.

Esta ecuación puede ser escrita como:

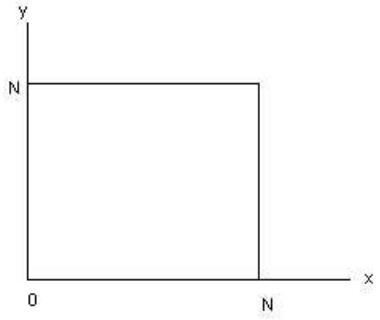

Figura 7–1: Experimento 1

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x, y)$$

Discretizando cada operador diferencial, utilizando diferencias finitas, se tiene:

$$-\frac{U_{i-1,j} - 2U_{i,j} + U_{i+1,j}}{h_x^2} - \frac{U_{i,j-1} - 2U_{i,j} + U_{i,j+1}}{h_y^2} = f_{i,j}$$

Con $i, j = 1, ..., N-1$, Lo que produce el siguiente sistema:

$$\begin{pmatrix} A_N & -I_N & & & & \\ -I_N & A_N & -I_N & & & \\ \vdots & & \ddots & & & \\ & & & -I_N & A_N & -I_N \\ & & & & -I_N & A_N \end{pmatrix} \begin{pmatrix} U_1 \\ U_2 \\ \vdots \\ U_{N-2} \\ U_{N-1} \end{pmatrix} = h^2 \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \\ 1 \end{pmatrix}$$

Donde I_N, es la matriz identidad de $N \times N$ y A_N, es la matriz:

$$\begin{pmatrix} 4 & -1 & & & & \\ -1 & 4 & -1 & & & \\ \vdots & & \ddots & & & \\ & & & -1 & 4 & -1 \\ & & & & -1 & 4 \end{pmatrix}$$

Primeramente veamos como se comporta el Multimalla algebraico cuando se varía el tamaño de malla, para el mismo problema con las anteriores condiciones de borde, los tamaños elegidos son: 10^4, 10^5 y 10^6 variables. Fijamos el suavizador al estandar, es decir, Gauss-Seidel, de la forma:

```
typedef Gauss_Seidel<Operator_CSR,double> TSmoother;
TSmoother* smoother = new TSmoother;
```

Primero para crear un tipo de datos correspondiente a este suavizador y luego una clase específica o instancia, con una estrategia de selección de malla gruesa simple (vease la sección 6.8.3), de la forma:

```
typedef simple_coarsening TCoarsening;
TCoarsening* C = new TCoarsening;
```

De forma similar al suavizador, especificamos el tipo y luego una instancia en concreto, los resultados obtenidos se aprecian en la figura 7–2.

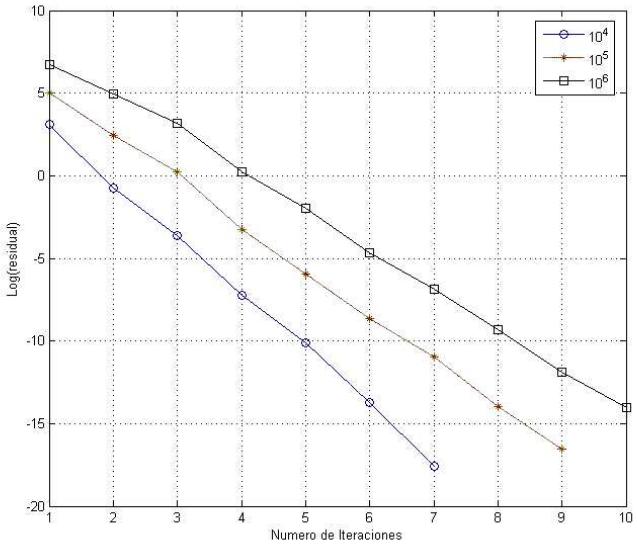

Figura 7–2: Historial del residuo para diferentes tamaños de malla.

Para ello se utilizó 2 iteraciones tanto en el pre-suavizador como en el post-suavizador y el clásico ciclo V, del multimalla. En la tabla 7–1, se muestra los detalles. Los tiempos, en dicha tabla así como en el resto del capítulo están expresados en segundos.

Tabla 7–1: Diferentes mallas en la fase de configuración del multimalla para la ecuación 7.1

Num. Var.	T. Conf.	T. Ejec.	Nro. Iter.	Num. Mallas
10^4	0.13	0.65	7	3
10^5	1.73	15.64	9	5
10^6	19.28	189.88	10	7

Mientras que el tiempo empleado en la fase de configuración y de resolución crece considerablemente, el número de iteraciones se incrementa levemente, lo cual es característico en este tipo de métodos, donde no existe dependencia del tamaño del problema, un caso particular se presenta cuando se emplea variantes al ciclo-V clásico del multimalla, tal como el ciclo-W, como se puede observar en la figura 7–3.

En esta situación en particular el número de iteraciones no varía aun cuando el tamaño del problema crece. En la tabla 7–2, se encuentran los tiempos registrados.

Tabla 7–2: Tiempos de ejecución y configuración para el multimalla algebraico, con gauss-seidel como suavizador y ciclo-W, para diferentes tamaños de malla

Num. var.	T. Conf.	T. Ejec.	Nro. Iter.	Num. Mallas
10^4	0.13	1.47	6	3
10^5	1.76	13.45	6	5
10^6	19.41	138.19	6	7

Según la tabla 7–2, se tiene mejor desempeño bajo esta configuración, que con la descrita en la tabla 7–1, especialmente con la malla más grande, el tiempo de ejecución resulta ser menor, porque se ejecuta solo 6 iteraciones del gradiente conjugado

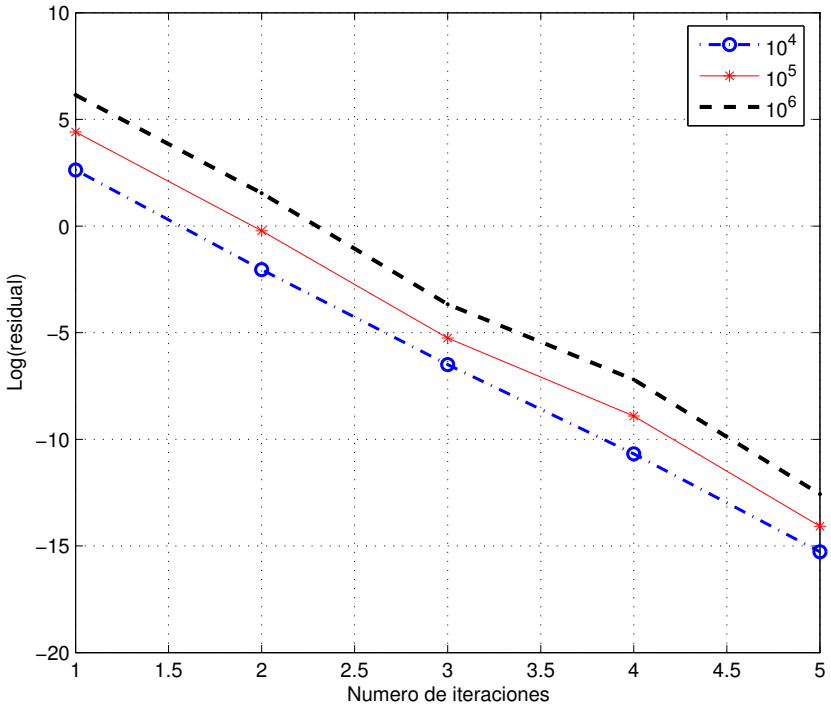

Figura 7–3: Historial del residuo para diferentes tamaños de malla.

aún cuando el número de iteraciones de los suavizadores se incrementa, esto debido a los ciclo-W del multimalla, pero se debe tener en cuenta que estas iteraciones de los suavizadores se realizarán en las mallas gruesas, por lo que resulta ser menos costoso, al menos para este tipo de aplicación. Por otro lado el tiempo de configuración es casi el mismo porque la estrategia de selección de malla gruesa es la misma.

Otro suavizador comunmente utilizado en los multimalla es Jacobi, se puede observar su desempeño, para el mismo problema, en la figura 7–4, tanto para el ciclo V como para el ciclo W. En el caso del ciclo V, el número de iteraciones para el pre y post-suavizador fueron de (0,3), respectivamente mientras que para el ciclo W fueron de (0,1), se pudo observar para este tipo de problema, estos valores tienen mejor desempeño, en general el número de pasos elegidos para el pre suavizado y post suavizado podria determinar el desempeño del multimalla incluso dependiendo

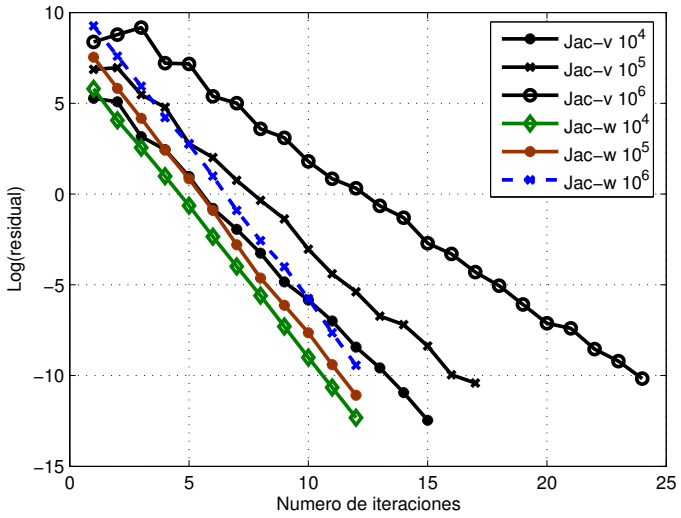

Figura 7–4: Historial del residual con Jacobi como suavizador con ciclos del multimalla v y w

del tipo de problema que se este resolviendo (como se verá más adelante).

Veamos como se comportan ambos suavizadores con diferentes estrategias de selección de malla gruesa, para ello fijaremos el tamaño de malla a 10^4, suavizadores Gauss-Seidel y Jacobi, estrategias de selección del tipo Ruge-Stüben y otro al azar, con el ciclo-V del multigrid, el número de pre y post iteraciones del suavizador a (2,2) respectivamente y para Jacobi con (2,2). Que se puede apreciar en la figura 7–5.

No se puede establecer con precisión el comportamiento de la estrategia de selección de malla gruesa aleatoria, pero para efectos de mostrar el comportamiento del método son útiles, con lo anterior, el comportamiento del multimalla dependerá mucho de la elección de los diferentes componentes, pero en especial del algoritmo de selección de malla gruesa, que es la parte que lo diferencia del multimalla geométrico y como vimos, degrada su desempeño, aún cuando elejimos un suavizador con mejores caracteristicas como es Gauss-Seidel.

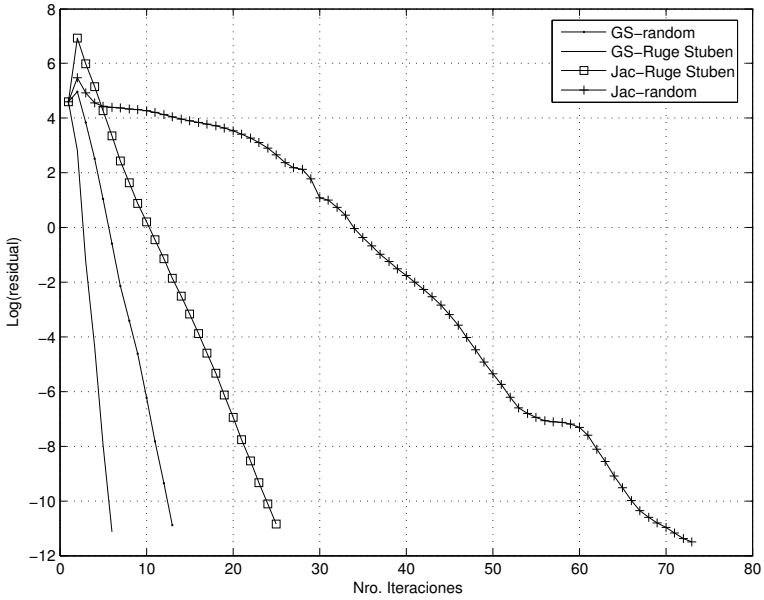

Figura 7–5: Historial del residuo para GS y Jacobi con RS y Aleatoria (Rand) como estraegias de selección de malla gruesa

Otras configuraciones respecto a la estrategia de selección de malla gruesa, el tipo de ciclo del multimalla, el suavizador y sus respectivos número de iteraciones, se pueden apreciar en la tabla 7–3.

Tabla 7–3: Comparación entre varias configuraciones para el multimalla algebraico

Suaviz.	T.Ejec.	Nro Iter.	Ciclo	N.Iter(pre,post)
Jacobi	6.11	108	V	(1,1)
Jacobi	4.99	75	V	(2,2)
Jacobi	4.67	61	V	(3,3)
Jacobi	21.09	98	W	(1,1)
Jacobi	15.80	68	W	(2,2)
Jacobi	13.95	56	W	(3,3)
GS	0.44	6	V	(1,1)
GS	0.57	6	V	(2,2)
GS	0.58	5	V	(3,3)
GS	1.21	5	W	(1,1)
GS	1.38	5	W	(2,2)
GS	1.25	4	W	(3,3)

En la tabla 7–3, la columna N.Iter, representa el número de pasos para el pre-uavizador y post-suavizador. Se puede observar que una configuración con Gauss-Seidel se tiene mejor desempeño tanto en número de iteraciones como en tiempo de ejecución o resolución. Por otro lado si utilizamos Jacobi como suavizador, se tiene mejor rendimiento en una configuración de ciclo-W del multimalla que en el caso de tres iteraciones para pre y post suavizado se obtiene una reducción de casi la mitad de itereaciones respecto a una configuración de un paso para pre y post suavizado, pero en cuanto a tiempo sucede algo inverso, obviamente por que se realiza más suavizado en el ciclo-W.

7.2 Experimento 2

Para este experimento el problema es similar al anterior pero con coeficientes discontinuos. Como se muestra en la ecuación 7.2, el propósito del experimento es mostrar el desempeño del multimalla en un dominio donde existen regiones con diferentes coeficientes y que estos reflejen discontinuidades, así mismo veremos el comportamiento para diferentes configuraciones de los valores de los coeficientes, como son 10^4, 10^6 y 10^8. El caso más simple cuando se definen solo dos regiones como en la figura 7–6.

Ecuación:

$$-\frac{\partial}{\partial x}(a\frac{\partial u}{\partial x}) - \frac{\partial}{\partial y}(b\frac{\partial u}{\partial y}) = f(x,y) \tag{7.2}$$

Con condiciones de borde del tipo Dirichlet: $u(x) = 0$, si $x \in \partial\Omega$, con $x \in [0,1] \times [0,1]$

Donde los coeficientes a y b, estan definidos como:

- En la región 1: $a = 1$ y $b = 1$
- En la región 2: $a = 10^C$ y $b = 1$

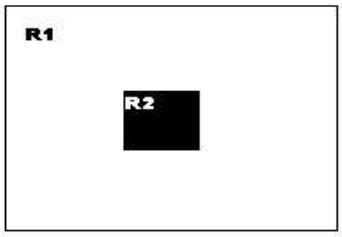

Figura 7–6: Dominio de la ecuación 7.2

Para el presente experimento tomaremos como $C = 4, 6, 8$. El tamaño de la región R2, es tomada como un decimo del tamaño de la region R1. Utilizando un esténcil de 5 puntos, se tiene la matriz esparcida del sistema con un patron de esparcidad similar al anterior problema. Discretizamos el operador:

$$\frac{\partial}{\partial x}(a\frac{\partial u}{\partial x})$$

en el punto i, de la forma:

$$\frac{a_{i-1/2}U_{i-1} - (a_{i-1/2} + a_{i+1/2})U_i + a_{i+1/2}U_{i+1}}{h^2}$$

Similarmente para:

$$\frac{\partial}{\partial y}(b\frac{\partial u}{\partial y})$$

Para una malla con 10^4 incógnitas, se puede apreciar la evolución del logaritmo del residuo en la figura 7–7, para los distintos valores de C, es decir, se tendrá los coeficientes para la ecuación 7.2, $a = 10^4, 10^6, 10^8$, con $b = 1$ fijo.

Del mismo modo podemos observar para una malla con 10^5 incógnitas, en la figura 7–8. En ambos casos el número de iteraciones para resolver se incrementa considerablemente conforme se varía el valor de los coeficientes.

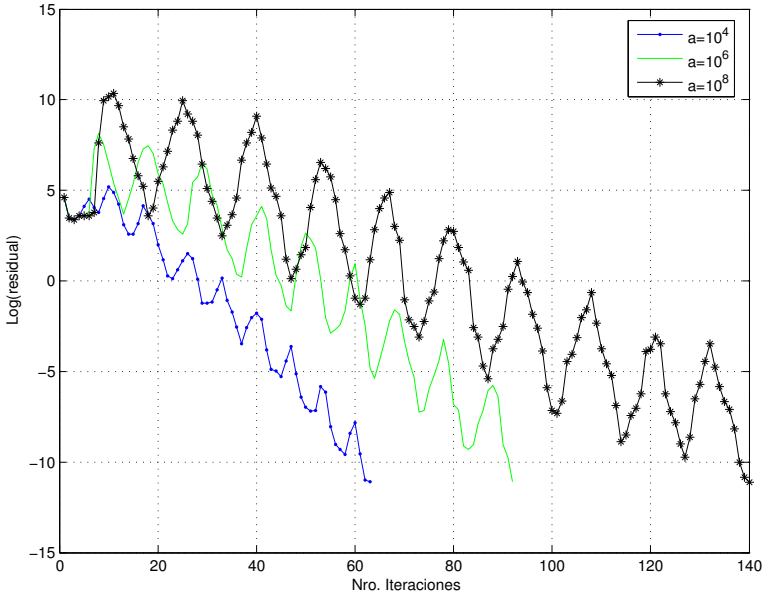

Figura 7–7: Evolución de los residuos con coeficientes discontinuos en la ec. 7.2, con 10000 incógnitas.

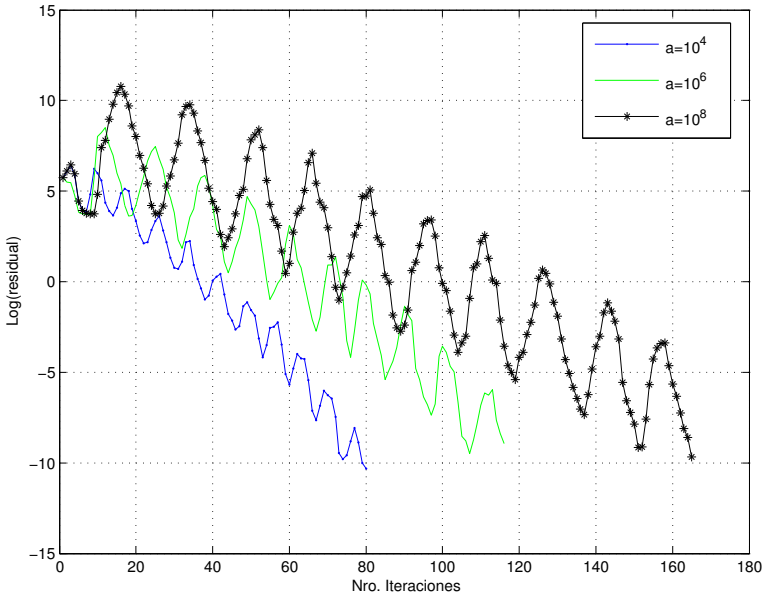

Figura 7–8: Evolución de los residuos con coeficientes discontinuos en la ec. 7.2, con 100000 incógnitas.

Los tiempos asociados a las figuras 7–7 y 7–8, se muestran en las tablas 7–4 y 7–5, realizando una comparación con el experimento anterior:

- En la fase de configuración los tiempos y el número de mallas generadas son casi las mismas para ambos experimentos, aquí la estrategia empleada es según *Ruge-Stüben*.
- Sin embargo si comparamos los tiempos de ejecución para el mismo tamaño de malla, igual suavizador y estrategia de selección de malla gruesa (Tablas 7–1 y 7–2) tanto el tiempo de resolución y el número de iteraciones empleados, crece notoriamente conforme se varía los coeficientes.

Tabla 7–4: Detalles de la ejecución para una malla de 10000 incógnitas con coeficientes discontinuos.

Coef.	T. Conf.	T. Ejec.	Nro. Iter.
$a = 10^4, b = 1$	0.14	5.69	64
$a = 10^6, b = 1$	0.15	9.68	92
$a = 10^8, b = 1$	0.15	11.62	131

Tabla 7–5: Detalles de la ejecución para una malla de 100000 incógnitas con coeficientes discontinuos.

Coef.	T. Conf.	T. Ejec.	Nro. Iter.
$a = 10^4, b = 1$	2.09	165.96	80
$a = 10^6, b = 1$	2.10	243.19	116
$a = 10^8, b = 1$	2.11	343.35	165

Tabla 7–6: Detalles de la ejecución para una malla de 100000 incógnitas con coeficientes discontinuos.

var. Gruesas	var. Finas	Total Incógnitas
5000	5000	10000
1275	3725	5000
323	952	1275

Si comparamos los resultados obtenidos en las tablas 7–4 y 7–5, el número de iteraciones para ambas mallas deberían ser cercanos (con la configuración de coeficientes correspondientes), pero sin embargo existe una considerable diferencia, esto se debe a que la implementación de la selección de malla gruesa no es exactamente la planteada por Ruge-Stüben en [49], ya que no se consideran una actualización de los pesos asociadas a las incógnitas (cuanto mayor sea el peso asociado a una variable mayor será la posibilidad de ser elegida como gruesa) para elegibilidad de variables gruesas, se debería realizar algun arreglo más sofisticado en la selección, de igual manera con la forma de realizar la interpolación.

En la tabla 7–6, se muestra los detalles de cada una de las mallas gruesas generadas en la fase de configuración para un total de incógnitas de 10^4, de la misma forma se puede extender a la de 10^5 incógnitas.

7.3 Experimento 3

En este experimento en lugar de una sola región donde los coeficientes son discontinuos, se tiene cuatro como se puede apreciar en la figura 7–9. En este caso los coeficientes fueron tomados como $a = 10^3$ y $b = 10^{-3}$. El propósito del experimento es mostrar como se degrada el desempeño del multimalla conforme se incrementan regiones con coeficientes discontinuos.

En la figura 7–10 se muestra una comparación del comportamiento de los residuales para mallas con 10^4, 10^5 y 10^6 variables, la configuración empleada para el multimalla algebraico fue con Gauss-Seidel como Suavizador y estrategia de selección de malla gruesa según Ruge-Stüben, los detalles se tienen en la tabla 7–7.

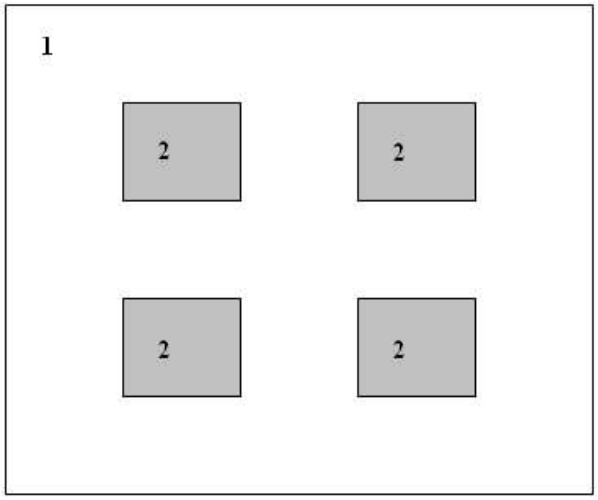

Figura 7–9: Dominio de la ecuación 7.2

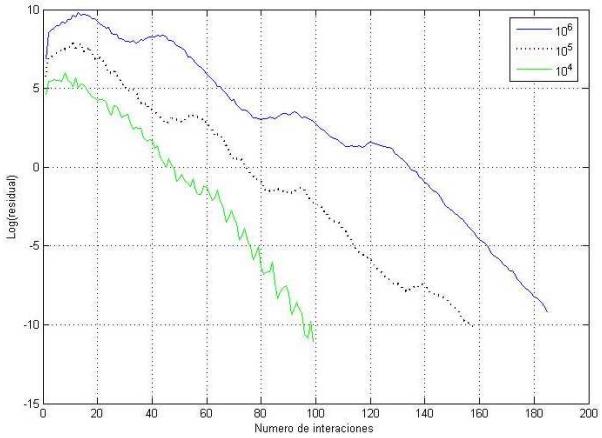

Figura 7–10: Dominio de la ecuación 7.2

Tabla 7–7: Tiempos de ejecución y configuración para la ecuación 7.2, según el dominio de la figura 7–10

Num. var.	T. Conf.	T. Ejec.	Nro. Iter.	Num. Mallas
10^4	0.1372	10.4644	99	3
10^5	1.77264	231.8589	158	5
10^6	19.0693	1401.9105	185	7

Como podemos observar, la ecuación es algo parecida a la del problema anterior en este caso la presencia de coeficientes discontinuos produce un incremento notable en el número de iteraciones y por consiguiente el tiempo empleado para resolver, en cuanto al tiempo para selección de malla gruesa es cercano al problema anterior, porque obviamente se realiza sobre las mismas variables aunque estas tengan coeficientes discontinuos no degrada el desempeño en la selección de variables gruesas, si se considera el algoritmo de Ruge-Stüben se esperaria que el número de variables gruesas seleccionadas debería ser menor, porque se realiza una discriminación de acuerdo a la condición establecida en la sección 3.4.2 (selección de malla gruesa), donde se toman en cuenta a las variables que sean mayores o iguales en valor absoluto al máximo de los vecinos una variable dada, para mayores detalles vease [49].

Veamos a más detalle como se comportan los algoritmos de selección, fijemos el tamaño de la malla en 10^4, con los mismas estrategias de selección de malla gruesa igual que el experimento anterior.

Tabla 7–8: Selección de malla gruesa para 10^4 de la ecuación 7.2, estrategia aleatoria

Gruesas	Finas	Total incógnitas
3699	6301	10000
730	2969	3699
136	594	730
26	110	136

Tabla 7–9: Selección de malla gruesa para 10^4 de la ecuación 7.2, estrategia Simple

Gruesas	Finas	Total incógnitas
5000	5000	10000
1275	3725	5000
325	950	1275
81	244	325

Tabla 7–10: Selección de malla gruesa para 10^4 de la ecuación 7.2, estrategia Ruge-Stüben

Gruesas	Finas	Total incógnitas
5000	5000	10000
1235	3765	5000
277	958	1235
63	214	277

En las tablas 7–8, 7–9 y 7–10, se tienen la selección realizada por cada estrategia, en el caso de la estrategia aleatoria se tiene un buen desempeño, para el ejemplo, lo cual no garantiza que en para todos los casos se logre el mismo resultado, es más por motivos descriptivos que se considera en el presente capítulo, cabe señalar que no toma en cuenta ningun criterio para seleccionar, podria ser que seleccione variables con menor número de conexiones y la idea del algoritmo de selección de malla gruesa es tratar de seleccionar aquellas variables con mayor número de conexiones o vecinos.

Por otro lado se debe tomar un compromiso en balancear número de variables gruesas y la interpolación, ya que si se tiene pocas variables gruesas podria ser que no se tenga una buena interpolación entre la malla gruesa y la fina, también dependiendo como se realize, si es directa o estandar o indirecta, (vease sección 3.4.1 para mayores detalles), por ello no basta con tener un número pequeño de variables gruesas sino también en como se transferiran los errores desde mallas gruesas hasta las finas. En cuanto a los tiempos, se muestran en la tabla 7–11.

Tabla 7–11: Tiempos de ejecución y configuración para la ecuación 7.2, con diferentes estrategias de selección de malla gruesa, para una malla con 10^4 variables.

Num. var.	T. Conf.	T. Ejec.	Nro. Iter.	Num. Mallas
Random	0.23	9.85	106	4
Simple	0.13	1.32	97	4
$Ruge - St\ddot{u}ben$	0.15	1.28	99	4

El tiempo empleado en el algoritmo según Ruge-Stüben, demanda de más tiempo aunque en el ejemplo no se puede apreciar o diferenciar considerablemente, debido al número de variables, pero uno deberia esperar que este consuma mayor tiempo, debido a que realiza un ordenamiento para seleccionar en primer lugar variables que tienen mayor número de conexiones, y así en forma descendiente, no obstante se debe señalar que no es una implementación completa de la propuesta de Ruge-Stüben, porque no se considera una actualización de los pesos (de las variables no seleccionadas aún) después de haber eligido una variable gruesa [49]. Por otro lado para el problema que se resuelve y el esténcil empleado, no se puede apreciar de forma más clara el comportamiento y las bondades de este algoritmo.

La evolución de los residuos conforme las iteraciones realizadas por el gradiente conjugado precondicionado con el multimalla se muestran en la figura 7–11. El suavizador usado en este caso es el Gauss-Seidel. En el gráfico podemos notar que con la estrategia de Ruge-Stüben consume más tiempo y emplea más iteraciones, esto se debe a que los pesos asignados a los nodos del operador de interpación se tomo como el promedio de todas las variables gruesas para la i-ésima variable de una malla dada, y al seleccionar menos variables que la estrategia simple se tiene menos aproximación que con la otra estrategia que selecciona mayor número de variables. Vease [46, 49], donde muestran otras formas de calcular el peso asignado a los nodos del operador de interpolación.

En este experimento no consideramos el método de Jacobi como suavizador, debido a un pobre desempeño, para el mismo problema y las mismas condiciones se registro al rededor de 700 a 900 iteraciones, lo cual representa demasiada diferencia respecto a Gauss-Seidel, se podria considerar algunas variantes y mejoras a Jacobi

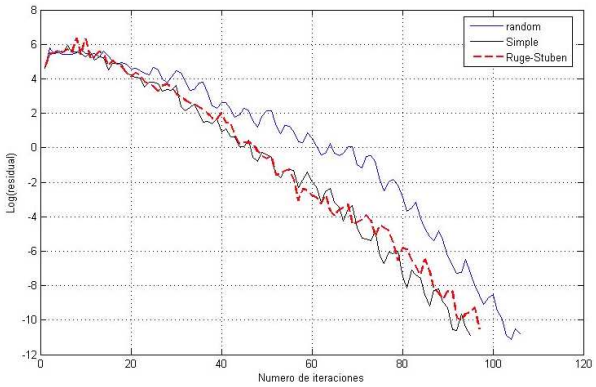

Figura 7–11: Historial del residuo para varios tipos de seleccíon de malla gruesa de la ecuación 7.2, para un tamaño de malla de 10^4 variables.

como es w-Jacobi, etc. para mejorar su rendimiento.

7.4 Experimento 4

En este experimento extenderemos la utilización de AMGLib++, a la ecuación desarrollada anteriormente, pero en un contexto más aplicativo, utilizaremos la ecuación de difusión anisotropica aplicado a un problema de procesamiento de imagenes. En este caso los coeficientes de difusión estan definidos de otra forma.

$$\frac{\partial u}{\partial t} = div(g(|Du|^2)Du)$$

Esta ecuación describe como se menciono la difusión anisotrópica no lineal, donde el papel de la función g, que representa los coeficientes de difusividad, estos están vinculados a la presencia de bordes, dentro de la imagen. Lo se que desea, es realizar difusión pero respetando o conservando los bordes de los objetos contenidos en la imagen, para mayores detalles teóricos sobre procesamiento de imágenes basados en EDPs véase [7], en el cual resolver esta ecuación es igual a resolver un problema

de procesamiento de imágenes, ya sea para remover ruido, resaltar o detectar bordes, etc. Uno de los primeros modelos presentados es el enfoque introducido por Perona-Malik [45], que es resuelto usualmente en forma explícita, utilizando diferencias finitas.

También se proponen implementaciones implícitas y semi-implícitas, en el presente experimento, se empleará una implementación semi-implícita. Existen versiones en el cual se resuelve el sistema no lineal mediante Newton, combinando con multimalla y de un tipo especial de multimalla denominado FAS [31]. Otros enfoques plantean particiones aditivas como Weickert [57] también están las basadas en wavelets, etc. y por supuesto basados en multimallas [6].

Consideremos la imagen inicial y la imagen con ruido gaussiano con media igual a 0.02, tal como se aprecia en la figura 7–12.

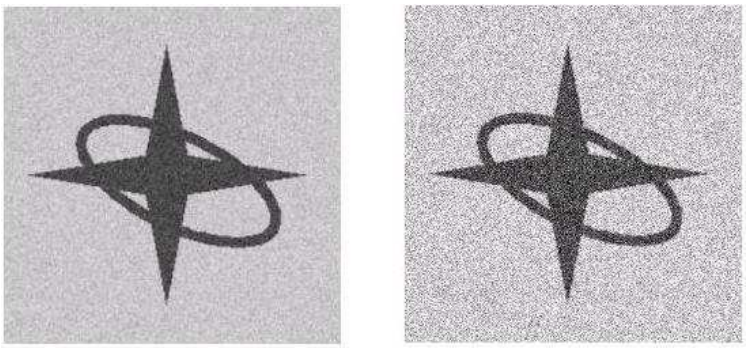

Figura 7–12: Imagen inicial, izquierda: original, derecha: con ruido Gaussiano de media 0.02

Para cada iteración (evolución en el tiempo de la solución de la ecuación), se realiza la construcción de las mallas, lo que representa un costo adicional, al proceso de resolver con multimallas algebraico, la ecuación de difusión anisotrópica, analicemos para el caso de solo 3 iteraciones con un paso en tiempo (dt) igual a 5, la tolerancia elegida fue de 10^{-7}. El tamaño de la imagen es de 248×248, lo que

produce una matriz esparcida de 61504 variables, si consideramos un esténcil de 5 puntos.

Se tiene el comportamiento del logaritmo del residual como se muestra en la figura 7–13.

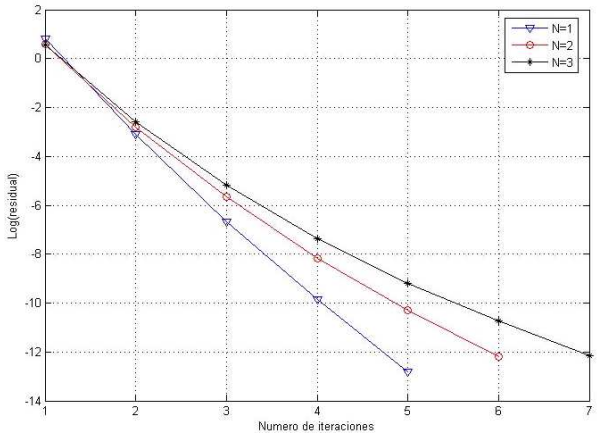

Figura 7–13: Historial del residuo para diferentes números de iteraciones en la ecuación de difusión anisotrópica

El resultado obtenido en solo 3 pasos es una versión, con una aceptable eliminación del ruido inicial incorporado, como se aprecia en la figura 7–14.

Para el caso de la figura 7–14, en la tabla 7–12, se puede resumir algunos resultados.

Si incrementamos las iteraciones se puede ver el comportamiento como se muestra en la figura, para 10 iteraciones 7–15.

Se pudo notar que conforme se avanza en tiempo se requiere más iteraciones del gradiente conjugado precondicionado, lo mismo que la complejidad del operador en

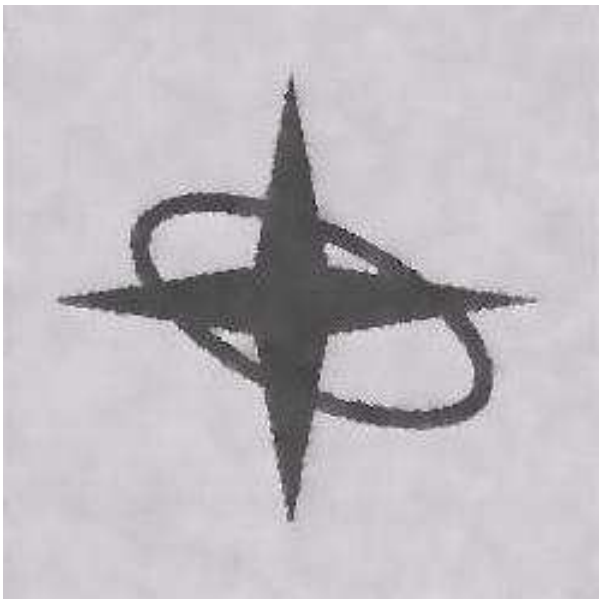

Figura 7–14: Imagen después de aplicar difusión anisotrópica en N=3, con dt=5

Tabla 7–12: Tiempos de ejecución y configuración para la ecuación de difusión anisotrópica

Espacio-Escala	T.Conf..	Iteraciones PCG	T.PCG
1	1.0025	5	4.4560
2	1.0884	6	5.6569
3	1.1393	7	6.6798
Total	3.2302	18	16.7927

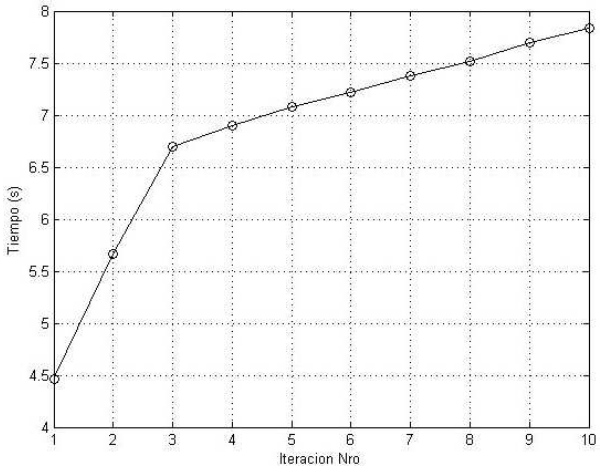

Figura 7–15: Tiempo de resolución de la ecuación de difusión anisotrópica con N=10, con dt=5

la generación de las mallas o fase de configuración del multimalla, es decir cuanta mas difusión se aplique más costoso será la aplicación del multimalla algebraico.

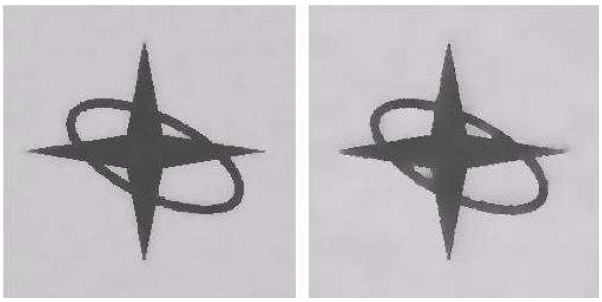

Figura 7–16: Resultados de aplicar difusión anisotrópica, izquierda: esquema explícito: dt=0.2 y escala espacio de 400, derecha: semi-implícito con dt=5 y escala espacio de 10

Esta técnica también puede ser utilizada para imágenes a color, tomando por separado cada canal, para el caso se tiene una imagen en formato RGB, por tanto se tiene 3 canales. Analizando el canal R, se tiene el comportamiento para cada iteración o evolución en el tiempo, tal como se muestra en la figura 7–17.

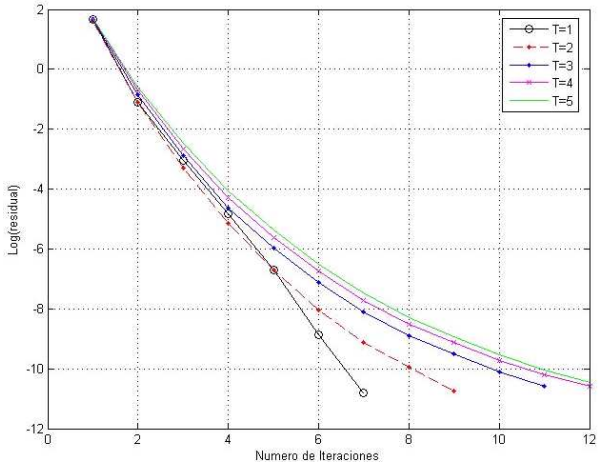

Figura 7–17: Resultados de aplicar difusión anisotrópica hasta una escala-espacio de 5, para Lena

Conforme se va suavizando se requieren más iteraciones del Gradiente Conjugado, por tanto el tiempo también se incrementa, la imagen inicial y los resultados obtenidos de Lena una vez procesados cada canal y luego ensamblados se muestran en las figuras 7–18 y 7–19.

Para poder observar de mejor forma el resultado de la difusión anisotrópica, veamos una sección de la imagen de Lena, como se aprecia en la figura 7–20, al cual se le aplicó difusión anisotrópica. En procesamiento de imágenes, una imagen es una función de R^2 en R, que representa la intensidad de color, para el caso consideremos la imagen en tonalidades grises, asi, el máximo es 1, que representa al blanco o máxima intensidad de luz, y cero representa al menos intenso, podemos observar el resultado en terminos de las superficies que representan ambas imágenes, antes y después de aplicar difusión anisotrópica.

En las figuras 7–21 y 7–22, se puede notar el resultado de la aplicación del operador de difusión anisotrópica, podemos observar que lo detectado como borde

Figura 7–18: Imagen Inician de Lena en RGG

se preserva mientras que las regiones que son homogeneas, quedan suavizadas uniformemente.

Figura 7–19: Resultado de aplicar difusión anisotrópica a 7–18, con escala-espacio de 5 y dt=5

Figura 7–20: Sección de lena antes y después de aplicar difusión anisotrópica, izqquierda antes, derecha después, con escala-espacio de 5 y dt=5

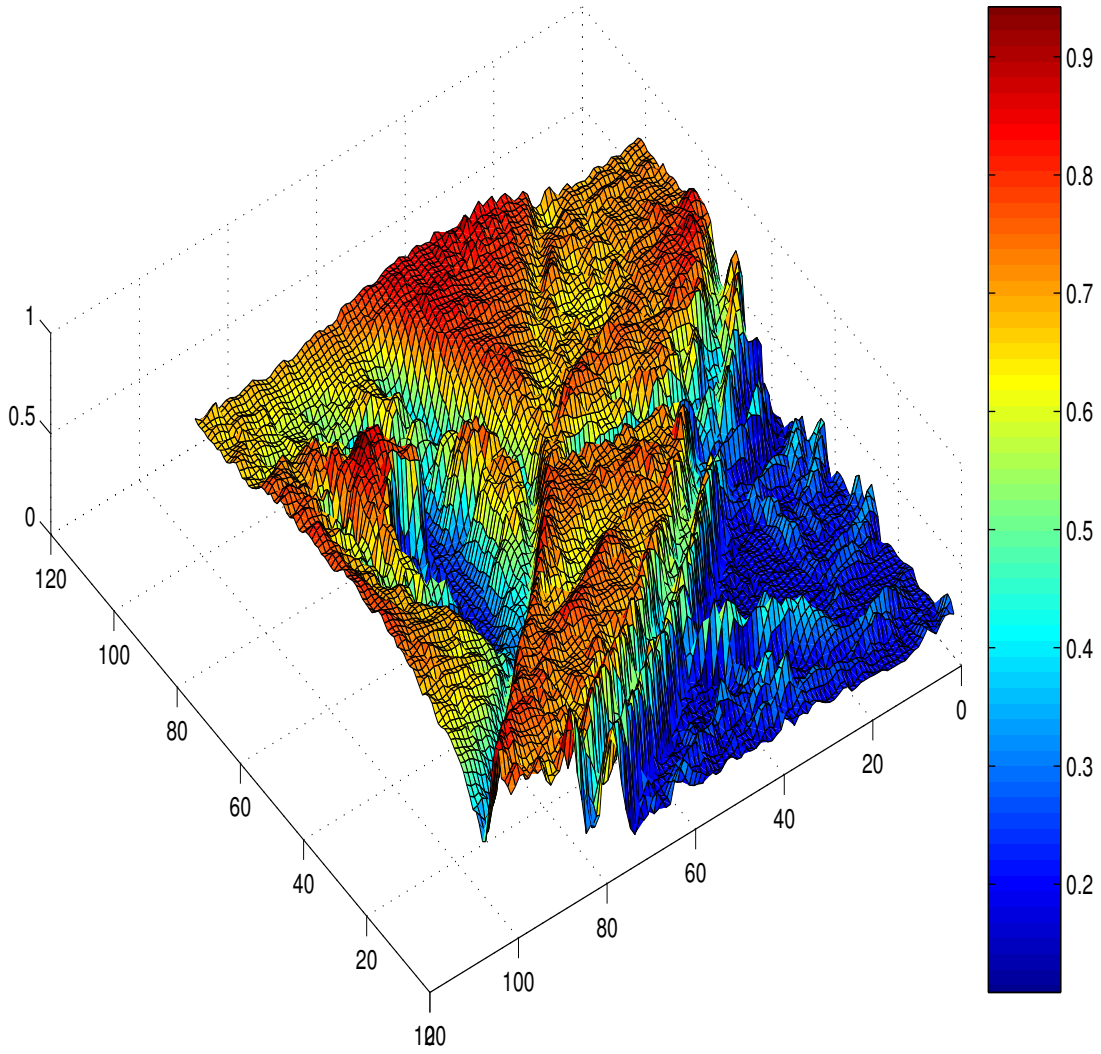

Figura 7–21: Superficie de la sección de Lena, antes de aplicar difusión anisotrópica.

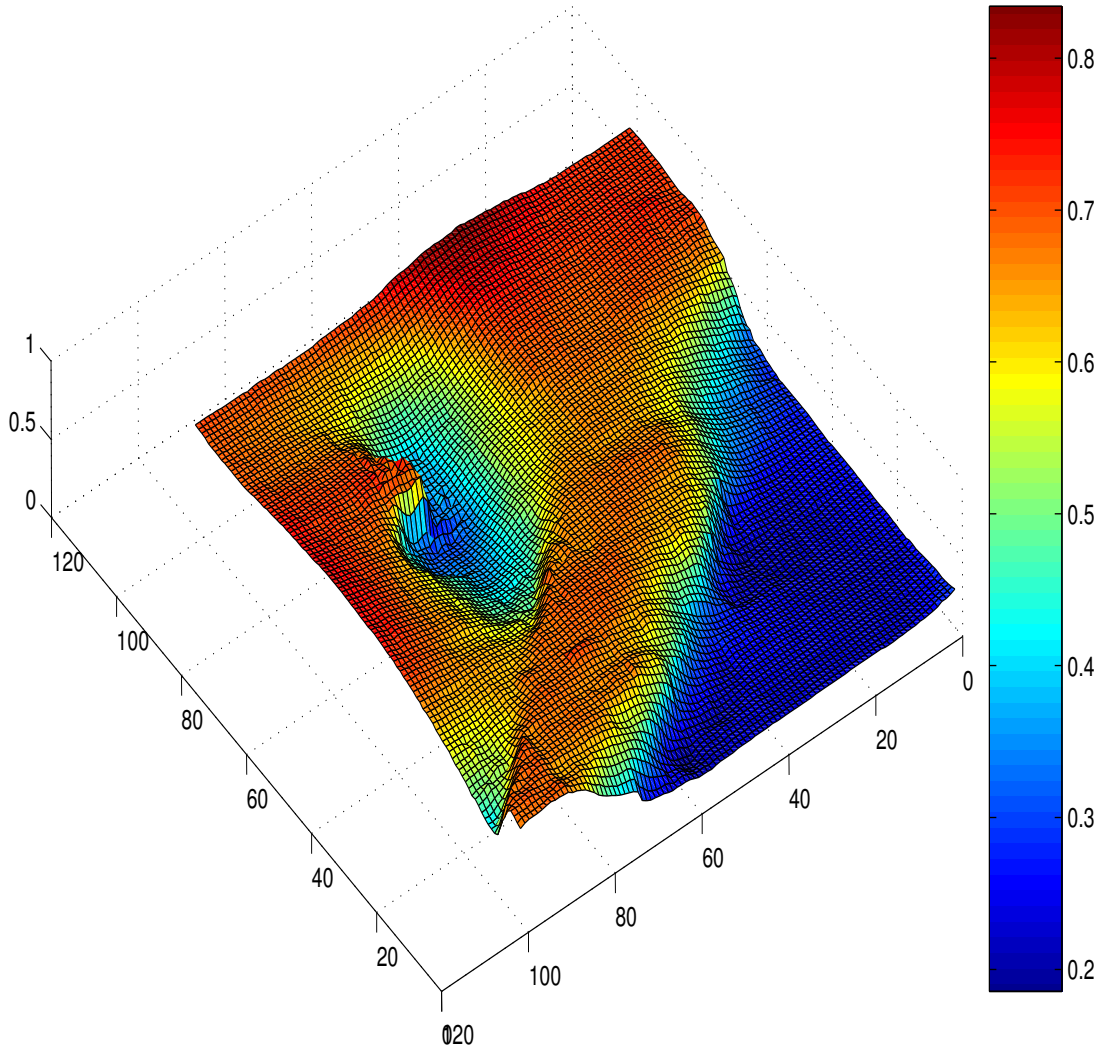

Figura 7–22: Superficie de la sección de Lena después de aplicar difusión anisotrópica, con escala-espacio de 5 y dt=5

CAPITULO 8
CONCLUSIONES Y TRABAJOS FUTUROS

Los multimallas son métodos de resolución de ecuaciones diferenciales parciales eficientes, en tiempo y en espacio, es una técnica que se vienen incorporando como parte de las aplicaciones en distintas áreas, tanto para aplicaciones científicas como para aplicaciones de ingeniería. Se puede diferenciar dos variantes de estos métodos, los geométricos, que trabajan en una seria de mallas generadas previamente, mientras que la versión algebraica resulta ser una evolución de los geométricos, al incorporarse el operador de Galerkin en y la interpolación dependiente del operador, resultando un enfoque algebraico basado puramente en la matriz de ecuaciones de discretización. Estos logran superar las dificultades de los geométricos en mallas con geometria complicada y la presencia de coeficientes discontinuos.

La programación genérica es adoptada como una forma de implementar librerias, para cómputo científico que permite trabajar con tipos paramétricos, conocido tambien bajo el nombre de programación politípica, porque esta orientado para aplicaciones que soporten cualquier tipo de datos, cuidando de que se implementen las interfaces entre las funciones y tales tipos. No esta restringido solo para la programación orientada a objetos, pero combinada con esta, brinda mucha mayor potencia como herramienta de implementación de librerias.

Las matrices resultantes de la discretización de una ecuación diferencia parcial, son del tipo esparcidos, donde una gran cantidad de entradas son ceros, la forma

en como se implementen puede determinar la eficiencia de la librería, existen gran variedad de formatos y/o representaciones, las que deben ser elegidos dependiendo del problema que se quiera resolver.

Con las consideraciones anteriores podemos concluir que AMGLib++ es una librería diseñada como un *Entorno Computacional Abstracto Orientado a Objetos* para resolver grandes sistemas lineales esparcidos, los que provienen de la discretización de ecuaciones diferenciales, utilizando alguna técnica de discretización. Como bondades de esta libreria se tiene la flexibilidad y alta personalización que se le brinda al usuario, para agregar y/o modificar componentes computacionales del método de multimalla algebraico y que sea de forma natural y sencilla.

AMGLib++ es la versión inicial de una librería más robusta, en el presente trabajo está concebido para computo no paralelo, sin embargo para más adelante se extenderá a la implementación de mecanismos que permitan incluir esta característica, en los diferentes niveles, tales como algoritmos de computo en paralelo relacionadas a operaciones de matrices y vectores, algoritmos de selección de malla gruesa en paralelo asi como la implementación de suavizadores y el mismo algoritmo del multimalla algebraico en paralelo. La adaptabilidad en terminos del mallado del dominio brinda mayores ventajas para ciertos problemas, por lo que se contemplará la implementacion de los aspectos mencionados, es el camino que se seguirá para futuras versiones de la librería. Lo mismo que se presentará mayor implementación de otros tipos de selección de malla gruesa y suavizadores.

APENDICES

APENDICE A
LAPACK

Es una librería escrita en fortra 77, desarrollada a finales de la decada de los ochenta, que contiene numerosas rutinas de algebra lineal, estas librerías han sido optimizadas razón por la cual son eficientes. En AMGLib++ se hace uso de las rutinas relacionadas con el manejo de vectores, tales como producto interior, operaciones de suma o resta, entre otros. Esta librería ha sido desarrollada considerando: Rendimiento, reusabilidad y portabilidad, tratando de optimizar las operaciones básicas en niveles y operaciones por bloques.

El primero de estos constituido por BLAS (Basic Linear Algebra Subprograms), en el nivel L1, estan las operaciones entre vectores, en el siguiente L2, se encuentran operaciones con matrices y vectores, finalmente en el ultimo nivel L3, estan las rutinas entre matrices.

La segunda consideración es hecha con el fin de aprovechar la arquitectura de los computadores, específicamente con las memorias que se encuentran cercanas al procesador denominadas *Cache*.

Las rutinas utilizadas de esta libreria son:

1. *DNRM2 (n, x, incx)*, calcula la norma euclideana, del vector x, de dimensión n.

2. *DAXPY(n, alpha, x, incx, y, incy)*, calcula $y = \alpha x + y$, con x, y vectores de dimension n, y α como escalar.
3. *DOT(n, x,incx, y, incy)*, Retorna el producto interno de vectores x, y, de dimensión n.
4. *DCOPY (n, x, incx, y, incy)*, Copia el vector x en y.
5. *DGESV(N, NRHS, A, LDA, IPIV, B, LDB, INFO)*, Calcula la solución del sistema $AX = B$, con NHR, como el número de vectores de lado derecho, A, la matriz del sistema; LDA, la dimensión del sistema, PIV, variable de salida, que indica los índices de los pivotes, B, son los vectores de lado derecho y también la solucion del sistema; LDB, la dimensión de B; INFO, devuelve un indicador del exito de la operación.

En todas estas rutinas los parametros *incx* y *incy*, se refieren al incremento en las posiciones de los arreglos: x e y. Para mayores detalles puede verse los manuales de usuario de esta libreria.

.

APENDICE B
EJEMPLO DE USO DE AMGLIB++

En este apéndice mostraremos un ejemplo de utilización de AMGLib++, el ejemplo en mención está orientado para resolver un problema cuyos coeficientes de la matriz son cargados desde un archivo, el lado derecho de la ecuación y el valor inicial iguales a cero. El objetivo de este anexo es mostrar los detalles claves y prácticos en la utilización de la libreria.

En primer lugar describiremos la estructura de directorios, de esto modo, saber donde se encuentran los archivos y el contenido de los mismos.

```
AMGLib++\include
        \source
        \lib
```

En el directorio "include" se encuentran los archivos:

- amg.h, contiene la implementación de la plantilla del método de multimalla algebraico y todo lo representado en el capítulo 6 como ámbito del AMG.
- coarsening.h, contiene lo especificado como ámbito de COARSENING.
- transfer.h, la plantilla para el ámbito de TRANSFER OPERATOR.
- solver.h, la plantilla para un método de resolución de sistemas lineales de la forma $Ax = b$, pero como clase abstracta.
- jacobi.h, una clase derivada de "solver" para el método de Jacobi.
- gauss_seidel.h, como la clase anterior pero para el método de Gauss-Seidel.
- pcg.h, otra clase derivada de "solver" para el método de gradiente conjugado precondicionado.

- gaussian.h, otra clase derivada de "solver" para el método de eliminación Gaussiana.
- operator_csr.h, se encuentra la declaración de la clase que implementa matrices esparcidas en formato de filas comprimidas.
- rg_coarsening.h, se encuentra la declaración de la clase que implementa una estrategia de selección de malla gruesa basada en lo planteado por Ruge-Stüben.
- rnd_coarsening.h, se encuentra la declaración de la clase que implementa una estrategia de selección de malla gruesa aleatoria.
- lapack.h, las rutinas de interface para usar las librerias LAPACK (vease anexo A).
- global.h, algunas variables y constantes de uso global en la libreria.

En el directorio "source", se encuentran los archivos:

- operator_csr.cpp, la implemtación de la clase para matrices esparcidas en formato CSR.
- rg_coarsening.cpp, la implemtación de la estrategia de selección de malla gruesa según Ruge-Stüben.
- rnd_coarsening.cpp, implementación de la estrategia de selección de malla gruesa aleatoria.
- test_amg.cpp, archivo principal donde se utilizará todo lo anterior para generar el archivo ejecutable (en C++ se conoce como el archivo que contiene la función "main").

Finalmente en relación con la estructura de directorios y archivos, se tiene al directorio lib, que contiene la libreria *libf2c.a*, necesario para utilizar las funciones de LAPACK.

Veamos ahora la construcción de una instancia de AMG, concretamente el archivo principal *test_amg.cpp*.

```
#include <iostream>
#include "global.h"
#include "operator_csr.h"
#include "global.h"
#include "rg_coarsening.h"
#include "transfer.h"
#include "gauss_seidel.h"
#include "gaussian.h"
#include "amg.h"
#include "pcg.h"

#define post 2  // Num of post-smoothing
#define pre 2   // Num of pre-smoothing
#define num_mgc 1 // multigrid V-Cycle
#define tol 1e-7 // stop criteria for PCG
#define max_iter 1000000 // max iteration for PCG
#define min_matrix 500 // min size of coarsest level


using namespace std;

int main(int argc, char* argv[])
{
  //-------------------------------------------------------------
  // Sparse matrix in CSR format
  //-------------------------------------------------------------
  Operator_CSR* A = new Operator_CSR;
  A->load_from_file(argv[1]);

  //-------------------------------------------------------------
  // Right-hand-side and initial value and result vector
  //-------------------------------------------------------------
  int n = A->get_num_nodes();
  double* u = new double[n];
  double* f = new double[n];
  double* u0 = new double[n];
  for ( int i =0; i<n; i++ )
  {
    u0[i] = 0;
    f[i] = 1;
    u[0] = 0;
  }

  //-------------------------------------------------------------
  //                  Smoother
```

```
//-------------------------------------------------------------
typedef Gauss_Seidel<Operator_CSR,double> TSmoother;
TSmoother* smoother = new TSmoother;
smoother->set_type_iteration( 3 ); // symmetric gauss-seidel

//-------------------------------------------------------------
//   Transfer operators (Interpolation & Restriction)
//-------------------------------------------------------------
typedef transfer_operator<TCoarsening,Operator_CSR> TTransfer;
TTransfer* T = new TTransfer;

//-------------------------------------------------------------
//       Coarsest Grid solver
//-------------------------------------------------------------
typedef Gaussian<Operator_CSR> TCoarse_Grid_Solver;
TCoarse_Grid_Solver* ge = new TCoarse_Grid_Solver;

//-------------------------------------------------------------
//       AMG setup
//-------------------------------------------------------------
typedef AMG<double,
            Operator_CSR,
            TCoarsening,
            TTransfer,
            TSmoother,
            TCoarse_Grid_Solver> TAMG;
TAMG* amg = new TAMG;

amg->set_min_size( min_matrix );
amg->set_coarse_grid_selector( C );
amg->set_transfer_operator( T );
amg->set_coarse_grid_solver( ge );
amg->set_matrix( A );
amg->set_presmoother( smoother );
amg->set_postmoother( smoother );
amg->set_num_pre_smoothing( pre );
amg->set_num_post_smoothing( post );
cout << "---------------------------------------" << endl;
cout << "Setup phase" << endl;
cout << "---------------------------------------" << endl;
amg->setup();

//-------------------------------------------------------------
//       AMG Preconditioner
//-------------------------------------------------------------
```

```
amg->set_right_hand_side( f );
amg->set_initial_guess( u0 );
amg->set_solution_vector( u );
amg->set_num_cycle( num_mgc );
amg->set_type_inc_smothing_step( false );

//-------------------------------------------------------------
//       PCG setup
//-------------------------------------------------------------
PCG<Operator_CSR,double, TAMG> pcg;
pcg.set_matrix( A );
pcg.set_preconditioner( amg );
pcg.set_initial_guess( u0 );
pcg.set_right_hand_side( f );
pcg.set_tolerance( tol );
pcg.set_max_iteration( max_iter );
pcg.set_solution_vector( u );

cout << "----------------------------------------" << endl;
cout << "Solving" << endl;
cout << "----------------------------------------" << endl;
pcg.solve();

//-------------------------------------------------------------
//       Save result vector
//-------------------------------------------------------------
ofstream ans( ans_file, ios::out );
if ( ans.fail() )
{
   cerr << "Open error, file: " << ans_file << endl ;
   exit( -1 );
}
for ( int i =0; i<n; i++ )
        ans << scientific
            << setprecision(16)
            << u[i] << endl;
ans.close();

//-------------------------------------------------------------
//       Save residual
//-------------------------------------------------------------
double* R = pcg.get_residual();
ofstream res( argv[2], ios::out );
if ( res.fail() )
{
```

```
     cerr << "Open error, file: " << res_file << endl ;
     exit( -1 );
  }

  for ( int i =0; i<pcg.get_iteration(); i++ )
    res << scientific
        << setprecision(16)
        << R[i] << endl;
  res.close();

  return 0;
}
```

Ahora veremos de cerca los dos métos principales en esta clase como son "setup" y "solve".

```
//--------------------------------------------------------------------
//      SETUP
//--------------------------------------------------------------------
template <typename VECTOR,
          typename OPERATOR,
          typename COARSE_GRID_SELECTOR,
          typename TRANSFER_OPERATOR,
          typename SMOOTHER,
          typename COARSE_GRID_SOLVER>
void  AMG<VECTOR,
        OPERATOR,
        COARSE_GRID_SELECTOR,
        TRANSFER_OPERATOR,
        SMOOTHER,
        COARSE_GRID_SOLVER>::setup()
{

  int i = 0;                    // initial level
  A_H.push_back( A_ );          // Fine level
  int op_size = A_H[0]->get_num_nodes();
  transfer_operator->set_coarsening( coarse_grid_selector );

  cout << "----------------------------------------" << endl;
  cout << "Coarse \t Fine \t Total" << endl;
  cout << "----------------------------------------" << endl;

  // get current time
  timeval InitialTime, EndTime;
```

```
  gettimeofday(&InitialTime, NULL);
  while ( op_size > min_size )
  {
    coarse_grid_selector->set_OPERATOR( A_H[i] );
    coarse_grid_selector->run();
    OPERATOR* p = transfer_operator->interpolator();
    OPERATOR* r = p->transpose();

    R.push_back( r );
    P.push_back( p );
    OPERATOR* a = mul( r, A_H[i], p );
    A_H.push_back( a );
    op_size = a->get_num_nodes();
    i++;
  }
  gettimeofday(&EndTime, NULL);
  cout << "setup time: "
       << ((EndTime.tv_sec*1e6+EndTime.tv_usec) -
          (InitialTime.tv_sec*1e6+InitialTime.tv_usec))*1e-6
       << " Sec." << endl;
  max_level = i;

}
```

Esta función representa a la fase de configuración del multimalla algebraico, como se puede notar se construyen los operadores de restricción (sentencia OPERATOR* r = p->transpose();) e interpolación (sentencia OPERATOR* p = transfer_operator->interpolator();), así como las mallas gruesas (sentencia OPERATOR* a = mul(r, A_H[i], p);). La declaración de la clase "AMG" ya se dio en la sección 6. La otra función importante *solve()*, solo contiene una invocación al método recursivo: *virtual void mgm(VECTOR* u, VECTOR* f, int level);* se conserva el método *solve()* para mantener la compatibilidad con la clase padre SOLVER. Por tanto veamos los detalles de la función *mgm(...)*.

```
//--------------------------------------------------------------------
//      MGM
//--------------------------------------------------------------------
template <typename VECTOR,
          typename OPERATOR,
```

```
          typename COARSE_GRID_SELECTOR,
          typename TRANSFER_OPERATOR,
          typename SMOOTHER,
          typename COARSE_GRID_SOLVER>
void AMG<VECTOR,
        OPERATOR,
        COARSE_GRID_SELECTOR,
        TRANSFER_OPERATOR,SMOOTHER,
        COARSE_GRID_SOLVER>::mgm( VECTOR* u, VECTOR* f, int level )
{
  if ( level == max_level )
  solve_coarse( A_H[max_level], u, f );
  else
  {
    smooth( A_H[level], u, f, 0 , level );
    VECTOR* d = new VECTOR[A_H[level]->get_num_nodes()];
    VECTOR* r = new VECTOR[R[level]->get_num_rows()];
    VECTOR* v = new VECTOR[R[level]->get_num_rows()];

    dcopy( A_H[level]->get_num_nodes(), f, d );
    mat_mul( A_H[level], u, d, -1, 1 );
    mat_mul( R[level], d, r, 1, 0 );

    for ( int i=0; i< R[level]->get_num_rows(); i++ )
      v[i] = 0;

    for ( int c=0; c<cycle; c++ )
    mgm( v, r, level + 1 );

    mat_mul( P[level], v, u, 1, 1 );
    smooth( A_H[level], u, f, 1, level );

    delete [] d;
    delete [] r;
    delete [] v;
  }
}
```

Lo cual es la representación del algoritmo de solución del multimalla, se hace uso de la función *mat_mul()*, para multiplicación de matriz por vector que debe ser compatible con la la clase que representa a las matrices y vectores. Por otro lado la función *smooth(...)* es la invocación de un método iterativo, para realizar suavizado,

el método iterativo está representado por las variables privadas de la clase AMG denominada *pre_smoother* y *post_smoother* del tipo abstracto SMOOTHER.

A continuación veamos el ámbito de TRANSFER contenido dentro del archivo *transfer.h*, la declaración de la clase *transfer_operator* construida como clase genérica o plantilla es:

```
#ifndef _TRANSFER_H_
#define _TRANSFER_H_

#include <iostream>
#include <stdlib.h>
#include <cassert>

using namespace std;

template <typename COARSE_GRID_SELECTOR,
          typename OPERATOR>
class transfer_operator
{
public:
  transfer_operator();
  virtual ~transfer_operator();
  void set_coarsening( COARSE_GRID_SELECTOR* S ) { selector = S; }
  virtual OPERATOR* restrictor();
  virtual OPERATOR* interpolator();
private:
  COARSE_GRID_SELECTOR* selector;
};
```

El métdo más importante en esta clase es *OPERATOR* interpolator()* que construye el operador de interpolación en base a la información que se extraerá de la clase *selector* que es un tipo paramétrico.

```
template <typename COARSE_GRID_SELECTOR, typename OPERATOR>
OPERATOR* transfer_operator<COARSE_GRID_SELECTOR,
                            OPERATOR>::interpolator()
{
  int            f;
```

```
  int            m;
  int            k;
  int            j;
  int            num_c;
  int            num_f;
  OPERATOR* P = new OPERATOR;

  num_c = selector->init_get_coarse();
  num_f = selector->init_get_fine();
  assert( num_c > 0 && num_f > 0 );

  P->resize( num_c + num_f, num_c );
  while (( f = selector->get_next_fine() ) >= 0 )
  {
    m = selector->init_get_interpolation_nodes( f );
    for ( k = 0; k <m; k++ )
    {
      j = selector->next_interpolation_node();
      P->set_value( f, j, 1.0/m );  // P(i,j)=1/m
    }
  }

  for ( int k = 0; k<num_c; k++ )
  {
    int i = selector->get_next_coarse();
    P->set_value( i, k, 1 );    // P(i,j) = 1
  }
  return P;
}
```

Donde *num_c* es el número de variables gruesas y *num_f*, el número de variables finas, la suma de estas dos debe ser igual al tamaño de la malla o la dimensión de la matriz g (variable privada de la clase *transfer_operator*). En el ciclo "WHILE" lo que se realiza es la inserción de valores en las entradas que correspondan a las variables finas (pesos de interpoalción) en el clase o variable P, que será el resultado de esta función. Finalmente en el ciclo "FOR" se insertan unos en las entradas correspondientes a las variables gruesas.

Ahora veremos el ámbito de COARSENING, contenido en el archivo *coarsening.h*, en este archivo se encuentra la declaración de la clase abstracta *coarsening*, que a su vez es una plantilla porque realizará la selección de malla gruesa sobre un tipo abstracto de datos denominado OPERATOR. Por ejemplo podría ser una clase para matrices esparcidas en formato de filas comprimidas (CSR).

```
#ifndef _coarsening_H_
#define _coarsening_H_

#include "global.h"

using namespace std;

// (index_fine, index_coarse), index_coarse => consecutive
typedef pair<int,int> coarse_node;
typedef set<int> set_int;

template<typename OPERATOR>
class coarsening
{
public:
  coarsening();
  virtual ~coarsening();
  void set_OPERATOR( OPERATOR* gg );
  virtual void run()=0;  // main method, run the coarsening process

  // interface methods for transfer operator
  virtual int init_get_coarse() = 0;
  virtual int init_get_fine() = 0;
  virtual int get_next_coarse() = 0;
  virtual int get_next_fine() = 0;
  virtual int init_get_interpolation_nodes( int f ) = 0;
  virtual int next_interpolation_node() = 0;

protected:

  // Abstract function, these are the interface with operator class
  virtual int extract()=0;
  virtual int delete_var( int v ) = 0;
  virtual void delete_neighbors( int v );
  virtual void add_coarse( int c ) = 0;
```

```
  virtual void add_fine( int f ) = 0;

  OPERATOR* g;   // The grid or level
private:

};
```

Una posible implementación del método principal "RUN" de esta clase podria ser como sigue:

```
void rnd_coarsening_graph::run()
{
  //--------------Clear old values
  var->clear();
  coarse->clear();
  fine->clear();
  interp_nodes->clear();

  int    m;
  int    c = -1;

  // Load undecided variables
  m = g->get_num_nodes();
  for ( int i=0; i<m; i++ )
    (*var)[i] = 0;

  // Split into coarse and fine
  while ( var->size() > 0 )
  {
    c = extract();
    delete_var( c );
    add_coarse( c );
    delete_neighbors( c );
  }
}
```

En esencia esta función lo que hace es seleccionar variables gruesas bajo *algún criterio*, el cual debe estar implementada en la función *extract()* y cuando se eliminan los nodos vecinos a la variable selecciónada, estos deben convertirse en variables finas.

En este apéndice se puede ver tanto la declaración de las clases que representan a cada ámbito definido en el capítulo 6, así como las principales funciones o métodos de clase, con ésto, la idea es mostrar como se realiza la interacción entre clases y sobre todo como se hacen uso de las funciones de las clases paramétricas.

REFERENCIAS BIBLIOGRÁFICAS

[1] *Algebraic multigrid methods for systems.* scai.fraunhofer.de/samg.0.html.

[2] *Los alamos algebraic multigrid code.* http://lanl.gov/Caesar/node13.html.

[3] *Multigrid net.* http://www.mgnet.org/.

[4] *Parallel and element based grey box linear equation solver.* http://www.numa.uni-linz.ac.at/Research/Projects/pebbles.html.

[5] *Parallel multigrid solver library for unstructured finite element problems.* http://www.columbia.edu/ma2325/promintro.html.

[6] S. Acton, *Multigrid anisotropic diffusion*, IEEE Transaction in Image Processing., 7 (1998), pp. 280–291.

[7] L. Alvarez, F. Guichard, P. Lions, and J. Morel, *Axioms and fundamentals equations of image processing*, Arch. Rational Mech. Anal., 123 (1993), pp. 199–257.

[8] D. G. A.Sain, *STL Tutorial and Reference Guide*, Addison-Wesley, second ed., 2001.

[9] O. Axelsson, *Iterative Solution Methods.*, Cambridge University Press., 1994.

[10] R. C. Backhouse and J. Gibbons, *Generic programming.*, Advanced Lectures Springer, (2003).

[11] R. C. Backhouse and P. F. Hoogendijk, *Generic properties of datatypes.*, Generic Programming., (2003), pp. 97–132.

[12] R. C. Backhouse, P. Jansson, J. Jeuring, and L. G. L. T. Meertens, *Generic programming: An introduction.*, Advanced Functional Programming., (1998), pp. 28–115.

[13] Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, and H. van der Vorst,

Templates for the Solution of Algebraic Eigenvalue Problems, Siam, 2000.

[14] R. Barrett, M.W.Berry, T. Chan, J. Demmel, J. Donato, J.Dongarra, R. P. V. Eijkhout, C. Romine, and H. van der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, Siam, second ed., 1994.

[15] A. Brandt, *Multi-level adaptive technique (mlat) for fast numerical solutions to boundary value problems.*, Notes in Physics, Springer-Verlag, 18 (1973), pp. 82–89.

[16] ——, *Multi-level adaptive solutions to boundary-value problems*, Math. Comp, 31 (1977), pp. 333–390.

[17] ——, *Algebraic multigrid theory: the symmetric case.*, Appl. Math. Comp, 19 (1986), pp. 23–56.

[18] E. Chow, *Parasails user's guide*, Lawrence Livermore National Laboratory, Tech. Report UCRL-MA, (2000), p. 137863.

[19] E. Chow., *A priori sparsity patterns for parallel sparse approximate inverse preconditioners.*, SIAM J. Sci. Comput., 21 (2000), pp. 1804–1822.

[20] E. Chow, *Parallel implementation and practical use of sparse approximate inverses with a priori sparsity patterns*, Int. J. High Perf. Comput. Appl., 15 (2001), pp. 56–74.

[21] P. de Zeeuw, *Matrix-dependent prolongations and restrictions in a black box multigrid solver.*, J.Comp.Appl.Math., 33 (1990), pp. 1–27.

[22] B. Eckel, *Thinking in C++*, Prentice Hall, 2002.

[23] R. D. Falgout, J. E. Jones, and U. M. Yang, *The desing and implementation of hypre, a library of parallel high performance preconditioners.*, LLNL Technical Report, (2004), pp. UCRL–JRNL–205459.

[24] ——, *Conceptual interfaces in hypre.*, Future Generation Computer Systems, Special Issue on PDE software, 22 (2005), pp. 239–251.

[25] R. D. Falgout and U. M. Yang, *hypre: a library of high performance preconditioners.*, Lecture Notes in Computer Science, Springer-Verlag, 2331 (2002), pp. 632–641.

[26] R. P. Fedorenko, *The speed of convergence of one iterative process.*, USSR, Comp. Math. And Math Phys., 4(3) (1964), pp. 227–335.

[27] G. Golub and C. V. Loan, *Matrix Computation.*, The Johns Hopkins University Press., second ed., 1989.

[28] W. Hackbusch, *Ein iteratives verfahren zur schnellen auflsung elliptischer randwertprobleme.*, Report 76-12 Universitat Koln, (1976).

[29] ——, *Iterative Solutions of Large Sparse Systems of Equations.*, Springer-Verlag., 1994.

[30] P. Hemker, *On the order of prolongations and restrictions in multigrid procedures.*, J.Comp.Appl.Math., 32 (1990), pp. 423–429.

[31] V. E. Henson, *Multigrid methods for nonlinear problems: An overview*, Lawrence Livermore National Laboratory Technical report., UCRL-JC-150259 (2002).

[32] E. Isaacson and H. B. Keller, *Analysis of Numerical Methods.*, Dover Publication., 1994.

[33] J. Jeuring and P. Jansson, *Advanced functional programming*, Lecture Notes in Computer Science-Springer-Verlag., 1129 (1996).

[34] J.W.Ruge and K.Stüben, *Algebraic multigrid*, S.F. McCormick(Ed) Multigrid Methods, SIAM, (1987).

[35] R. Kettler, *Analysis and comparison of relaxation schemes in robust multigrid and preconditioned conjugated methods.*, Springer, (1982), pp. 1–176.

[36] L.-Q. Lee and A. Lumsdaine, *Generic programming for high-performance scientific applications.*, Concurrency - Practice and Experience., 17 (2005), pp. 7–8,941–965.

[37] Z. Li, Y. Saad, and M. Sosonkina, *parms: a parallel version of algebraic recursive multilevel solver.*, J.Comp.Appl.Math., 33 (2001), pp. 1–27.

[38] S. B. Lippman, *Essential C++.*, Addison-Wesley., 2002.

[39] N. Mateev, K. Pingali, and P. Stodghill, *Next-generation generic programming and its application to sparse matrix computations.*, In Proceedings of the 2000 International Conference on Supercomputing,Santa Fe, New Mexico., (2000).

[40] A. B. S. F. McCormick and J. W. Ruge, *Algebraic multigrid (amg) for automatic multigrid solutions with application to geodetic computations.*, Report, Inst. for Computational Studies, Fort Collins, Colo, (1982).

[41] S. F. McCormick, *Algebraic multigrid (amg), in multigrid methods.*, Frontiers in Applied Mathematics, SIAM, 3 (1987), pp. 73–130.

[42] A. P. P. M. L. D. Musser, *The C++ Standard Template Library*, Prentice Hall, 2001.

[43] D. R. Musser and A. A. Stepanov, *Generic programming*, ISSAC., (1988), pp. 13–25.

[44] C. Oosterlee and T. Washio, *An evaluation of parallel multigrid as a solver and preconditioner for singulary perturbed problems.*, SIAM J. Sci. Comp., 19 (1998), pp. 87–110.

[45] P. Perona and J. Malik, *Scale space and edge detection using anisotropic diffusion*, IEEE Trans. Pattern Anal. Mach. Intell., 12 (1990), pp. 629–639.

[46] Y. Saad, *Iterative Methods for Sparse Linear Systems*, Siam, second ed., 2000.

[47] Y. Saad and M. Sosonkina, *Distributed schur complement techniques for general sparse linear systems*, SIAM Journal on Scientific Computing, 21 (1997), pp. 1337–1356.

[48] Y. Saad and B. Suchomel., *Arms : An algebraic recursive multilevel solver for general sparse linear systems.*, Technical Report umsi-99-107-Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, MN, 2001, revised version of umsi-99-107, (99).

[49] K. Stuben, *Algebraic multigrid (amg) an introduction applications.*, GMD-Report, 70 (1999).

[50] ——, *A review of algebraic multigrid.*, GMD-Report, 69 (1999).

[51] C. Tong and R. S. Tuminaro, *Ml 2.0 smoothed aggregation user's guide.*, Sandia National Laboratories., Technical Report (2002), pp. SAND2001–8028.

[52] L. N. Trefethen and D. Bau, *Numerical Linear Algebra.*, SIAM., 1997.

[53] O. C. Trottenberg U., Schuller A., *Multigrid*, Academic Press Inc.(London) Ltd., 2000.

[54] P. Vanel, J.Mandel, and M.Brezina, *Algebraic multigrid by smoothed aggregation for second and fourth order elliptic problems.*, Computing, 56 (1996), pp. 179–196.

[55] R. Varga, *Matrix Iterative Analysis.*, Prentice Hall., 1962.

[56] V.E.Henson and U.M.Yang, *Boomeramg: a parallel algebraic multigrid solver and preconditioner*, LLNL technical report, (2000), pp. UCRL–JC–141495.

[57] J. Weickert, B. ter Haar Romeny, and M. Viergever, *Efficient and reliable schemes for nonlinear diffusion filtering*, IEEE Transactions on Image Processing., 7 (1998), pp. 398–410.

[58] P. Wesseling, *An Introduction to Multigrid Methods*, John Wiley and Sons Ltd., 1992.