9/1994-95

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
UNIDAD DE TERCER CICLO Y POSTGRADO

Reunido el día de la fecha,el Tribunal nombrado por el Excmo.Sr.Rector Magfco. de esta Universidad, el aspirante expuso esta TESIS DOCTORAL.

Terminada la lectura y contestadas por el Doctorando las objeciones formuladas por los señores jueces del Tribunal,éste calificó dicho trabajo con la nota de APTO "CUM LAUDE"

Las Palmas de G. C.,a 23 de Noviembre de 1994.

El Presidente: Dr. D. Francisco Rubio Royo,

El Secretario: Dr. D. Pedro Damián Cuesta Moreno,

El Vocal: Dr. D. Roberto Moreno Díaz,

El Vocal: Dr. D. Luis Gavete Corvinos,

El Vocal: Dr. D. Alfredo Bermúdez de Castro,

El Doctorando: D. Manuel Jesús Galán Moreno,

Universidad de Las Palmas de Gran Canaria

Departamento de Matemáticas

Tesis Doctoral

"Avances en el Método del Residuo Mínimo Generalizado (GMRES), su desarrollo en ANSI-C con algoritmos de vectorización y paralelización y sus aplicaciones al M.E.F."

Presentada por: *D. Manuel J. Galán Moreno*

Dirigida por: *Dr. D. Gabriel Winter Althaus*

Dr. D. Gustavo Montero García

El Doctorando

Fdo. Manuel J. Galán Moreno

El Director

Fdo. Gabriel Winter Althaus

El Director

Fdo. Gustavo Montero García

Las Palmas de Gran Canaria a 21 de Septiembre de 1.994

para Elena, con cariño...

Agradecimientos:

Deseo expresar mi gratitud por la ayuda, colaboración y apoyo prestados en la realización de la presente Tesis Doctoral:

A mis Directores de Tesis: Prof. D. ***Gabriel Winter Althaus*** *y Dr. D.* ***Gustavo Montero García****, a quienes, sin duda, "robé" mucho tiempo de dedicación familiar.*

Al Dr. D. ***Pedro Almeida Benítez****, Tutor, que dirigió mis primeros pasos en la Universidad de Las Palmas de Gran Canaria.*

A los miembros del Centro de Investigación Interdepartamental de Aplicaciones Numéricas en la Ingeniería (CEANI) y en especial al Dr. D. ***Pedro D. Cuesta Moreno****, que me dedicó mucho de su tiempo y atención.*

A los miembros del Departamento de Matemáticas y a la Universidad de Las Palmas de Gran Canaria que hicieron posible, con su aportación de medios, y el apoyo que siempre recibí por su parte, la realización de la presente Tesis.

Finalmente deseo hacer un recuerdo en estas pobres líneas del estimado compañero y amigo que nos abandonó de forma inesperada el Profesor D. ***Miguel Galante Guille****.*

A todos ellos, mi más sincero agradecimiento.

Resumen de la Tesis:

La presente tesis doctoral titulada "Avances en el Método del Residuo Mínimo Generalizado (GMRES), su desarrollo en ANSI-C con algoritmos de vectorización y paralelización y sus aplicaciones al Método de los Elementos finitos", desarrolla una investigación íntimamente enraizada en el campo de la Matemática Aplicada.

Uno de los problemas actuales más importantes y con mayor repercusión en los campos de la Ciencia y de la Tecnología es la resolución de Ecuaciones en Derivadas Parciales por el Método de los Elementos Finitos.

Por la discretización del operador integral, aparecen grandes sistemas de ecuaciones con una característica muy importante: la mayoría de las entradas son nulas (matrices huecas).

En la presente Tesis se desarrollan varios puntos:

- Mejora substancial del conocido método iterativo de Residuos Mínimos Generalizado para la resolución de sistemas.

- Uso de varias técnicas de optimización de convergencia basadas en el uso de precondicionadores.

- Nuevo procedimiento de ensamblaje de las matrices de "rigidez".

- Nuevo esquema de almacenamiento de dicha matriz (almacenamiento hipercompacto) que no sólo minimiza el uso de memoria RAM sino que también permite la vectorización y paralelización de las operaciones.

- Desarrollo de varias aplicaciones "test", para comprobar la validez del método y la mejora substancial que representa, siendo posible la resolución de grandes problemas con herramientas informáticas no demasiado sofisticadas en tiempos razonables.

ÍNDICE

Índice

GENERALIDADES

INTRODUCCIÓN

La resolución por el método de elementos finitos de problemas de contorno estacionarios o evolutivos, formulados en ecuaciones en derivadas parciales, envuelve dos pasos:

En primer lugar el problema es formulado a partir del modelo matemático, en forma variacional. El segundo paso corresponde a la resolución numérica del problema variacional en dimensión finita.

En esta etapa, en primer lugar, se genera una adecuada discretización geométrica del dominio espacial.

En segundo lugar tenemos que, de acuerdo a la discretización funcional adoptada para la resolución del problema, construir la matriz "de rigidez" global del sistema por ensamblaje o suma de contribuciones de todos los elementos finitos del mallado.

Finalmente debemos abordar la resolución del sistema, siendo de interés el disponer de un método robusto de resolución de sistemas de ecuaciones.

Asimismo es de interés cuantificar el error en la solución obtenida, mediante métodos de estimación del mismo.

La primera parte de la resolución depende de la clase de problema: sus particularidades físicas, la formulación del modelo matemático, etc., no así la segunda parte que es más general en la resolución numérica.

En la segunda etapa, debemos tener en cuenta las siguientes consideraciones:

- Obtenida la matriz global de rigidez debemos acometer la resolución de sistemas de ecuaciones lineales de orden elevado y con un alto porcentaje de entradas nulas. Este efecto se hace notar de forma incluso mayor en la resolución de problemas en 3D.

- La distribución de las entradas no nulas es aleatoria cuando se utilizan mallados no estructurados y tras varias posibles etapas de refinamiento y desrefinamiento, o tras varias etapas de adaptación de la malla.

- Sistemas mal condicionados: Algunas veces las restricciones impuestas y los parámetros de la formulación producen sistemas con un "número de condicionamiento" elevado para la matriz del sistema.

- Adaptabilidad: Aunque nuestra implementación estará dirigida a la resolución de un problema concreto, debemos intentar hacerlo de la manera más portable a problemas similares, con el menor esfuerzo posible.

Todas estas consideraciones nos conducen a los siguientes planteamientos:

1. Desarrollo del código informático/computacional en ANSI C, siendo este lenguaje de programación portable, muy rápido y flexible para nuestra implementación.

2. Uso de un esquema de almacenamiento de matrices optimizado, que se conoce como esquema "hipercompacto".

3. Un método de ensamblaje de la matriz basado en subrutinas de cadena simplemente enlazada.

4. Elección de métodos de resolución iterativos potentes y robustos.

5. Contemplar la posibilidad de implementación vectorial-paralela, con el gran ahorro de tiempo que significa, y la posibilidad de aplicarse en futuros desarrollos tanto en "hardware" como en "software".

GENERALIDADES SOBRE MÉTODOS DE RESOLUCIÓN NUMÉRICA DE SISTEMAS DE ECUACIONES LINEALES.

Exponemos en este apartado, teniendo en cuenta el entorno operacional en el que se sitúa la temática de la presente Tesis, y a modo de introducción, una panorámica general y sucinta sobre la resolución de sistemas lineales, a fin de explicar los inconvenientes planteados por los métodos directos y, por ello, la necesidad del uso de los métodos iterativos en la resolución numérica.

Previamente se recuerdan algunas definiciones y nociones básicas con el fin de fijar la notación en posteriores desarrollos.

PRELIMINARES

Definición de norma matricial inducida por una norma vectorial y propiedades:

$$\|A\|_p = \max_{x\neq 0} \frac{\|Ax\|_p}{\|x\|_p} = \max_{\|x\|_p=1} \|Ax\|_p \quad p \geq 1$$

Las propiedades más importantes de dichas normas:

- $\|AB\|_p \leq \|A\|_p \|B\|_p \quad A \in \Re^{mxn} \quad B \in \Re^{nxq}$
- $\|A\|_1 = \max_{1\leq j\leq n} \sum_{i=1}^{m} |a_{ij}| \quad A \in \Re^{mxn}$ (máximo de la suma por columnas)
- $\|A\|_\infty = \max_{1\leq i\leq m} \sum_{j=1}^{n} |a_{ij}| \quad A \in \Re^{mxn}$ (máximo de la suma por filas)
- $\|A\|_2 = \sqrt{max\,\lambda_i} \qquad \lambda_i \in \{valores\ propios\ de\ (A^T A)\}$ (esta norma es invariante si se realiza una transformación ortogonal).

Se recuerdan las siguientes definiciones:

Radio espectral de una matriz cuadrada real como:

$$\rho(\mathbf{A}) = \max_{\lambda \in \{valorespropiosdeA\}} \{|\lambda|\}$$

Número de condicionamiento de una matriz cuadrada real:

$$\kappa_p(A) = \|A\|_p \|A^{-1}\|_p$$

Nociones sobre números en coma flotante

Cada operación llevada a cabo por el ordenador va acompañada de un error de redondeo debidos a los condicionantes físicos de estas máquinas que sólo son capaces de representar un subconjunto de los números reales. Este subconjunto es llamado números en coma flotante y queda determinado por: La base β, la precisión t, y el intervalo que define el rango del exponente [L,U].

El subconjunto F consiste en los números que tienen la forma:

$$f = \pm d_1 d_2 \ldots d_t \, x \beta^e \quad 0 \le d_i < \beta, \quad d_1 \neq 0, \quad L \le e \le U$$

También se añade el cero. Hay que tener en cuenta que para un valor no nulo f∈ F tenemos que:

$$m \le |F| \le M$$

donde $m = \beta^{L-1}, M = \beta^U \left(1 - \beta^{-t}\right)$.

En un modelo de aritmética de computador el conjunto G se define como:

$$G = \{x \in \Re \mid m \le |x| \le M\} \cup \{0\}$$

Y el operador fl(oat): G→F, donde fl es una aplicación de los números reales del subconjunto G en su representación computacional. El operador fl cumple que

$$fl(x) = x(1+\varepsilon) \qquad |\varepsilon| \le u \qquad x \in G$$

Donde "u" (precisión del computador) se define como $u = \frac{1}{2}\beta^{1-t}$.

Debido a esta representación se pueden producir perdidas extremas de dígitos significativos cuando se efectúan cómputos aditivos entre números de magnitudes muy dispares.

Métodos Directos

Algunos de estos métodos se usan como tales, o bien como precondicionadores (en forma modificada) en los métodos iterativos.

La idea esta basada en la eliminación. Sin embargo, usando este método en la práctica, vemos que este tiene algunos inconvenientes. En primer lugar, los errores de redondeo pueden falsear la solución final; en segundo lugar, pueden fallar incluso en problemas simples.

La solución puede cambiar si la matriz y los términos independientes sufren pequeñas variaciones. Esto conduce al uso de procesos más estables donde se introduce el "pivoteo". A continuación se dan las propiedades del proceso de eliminación Gaussiana.

El método de eliminación Gaussiana

Es el algoritmo de elección cuando la matriz A es cuadrada no singular, densa, y relativamente pequeña.

Cálculo de la descomposición LU

Se demuestra que se puede fabricar una transformación de Gauss M_1, M_{n-1} tal que la matriz U dada por

$$M_{n-1}\, M_{n-2} \ldots M_2\, M_1\, A = U$$

Sea triangular superior. En consecuencia el algoritmo para calcular las M_i supone que la transformación de Gauss esta determinada y es tal que:

$$A^{(k-1)} = M_{k-1} \ldots M_1\, A = \begin{bmatrix} A_{11}^{(k-1)} & A_{12}^{(k-1)} \\ 0 & A_{22}^{(k-1)} \\ k-1 & n-k+1 \end{bmatrix}^{k-1}_{n-k+1}$$

donde $A_{11}^{(k-1)}$ es triangular superior. Si

$$A_{22}^{(k-1)} = \begin{bmatrix} a_{kk}^{(k-1)} & \cdots & a_{kn}^{(k-1)} \\ \vdots & & \vdots \\ a_{nk}^{(k-1)} & \cdots & a_{nn}^{(k-1)} \end{bmatrix}$$

y $a_{kk}^{(k-1)}$ es distinto de cero, entonces los multiplicadores

$$l_{ik} = a_{ik}^{(k-1)} / a_{kk}^{(k-1)} \quad i = k+1 \ldots\ldots n$$

están definidos. Por tanto si $M_k = I - \alpha^{(k)} \varepsilon_k^T$ donde

$$a^{(k)} = (0, \ldots., 0, l_{k+1,k}, \ldots., l_{nk})^T.$$

entonces

$$A^{(k)} = M_k A^{(k-1)} = \begin{bmatrix} A_{11}^{(k)} & A_{12}^{(k)} \\ 0 & A_{22}^{(k)} \\ k & n-k \end{bmatrix}^{k}_{n-k}$$

con $A_{11}^{(k)}$ triangular superior. Esto ilustra el paso k-ésimo de la eliminación Gaussiana. Si los pivotes $a_{kk}^{(k-1)} \neq 0$ para $k = 1,....,n-1$ entonces

$$A^{(n-1)} = M_{n-1} \ldots M_1 A = U$$

La inversa de $M_{n-1} \ldots M_1$ puede ser dada por

$$L = (M_{n-1} \ldots M_1)^{-1} = \prod_{i=1}^{n-1} (I + \alpha^{(i)} e_i^T) = I + \sum_{i=1}^{n-1} \alpha^{(i)} e_i^T$$

La matriz es triangular inferior y la diagonal (L)=I. Esto implica A=LU.

Coste computacional de la eliminación Gaussiana en matrices densas:

- Eliminación Gaussiana $O(2n^3/3)$
- Sustitución progresiva $O(n^2)$
- Sustitución regresiva $O(n^2)$

Para poder evaluar el efecto que sobre la solución tiene la perturbación sobre las matrices A y b hacemos uso del número de condición (condicionamiento).

Según los teoremas demostrados para el análisis de estabilidad de la eliminación Gaussiana (ver Golub, Van Loan[24]), ésta puede producir de forma arbitraria resultados inexactos incluso en el caso de sistemas bien condicionados: el método puede ser inestable, dependiendo de la matriz.

Para evitar estos efectos indeseables se recurre con frecuencia a la técnica del "pivoteo", pudiendo ser total o parcial.

También se recurre, a veces, al llamado refinamiento iterativo, que no suele dar unos resultados mejores ya que la ventaja que presenta de utilizar menos espacio de almacenamiento (al utilizar una menor precisión inicial) queda anulada por la necesidad de conservar la matriz original, no siendo posible la factorización "in situ".

Cuando la matriz A es simétrica y definida positiva la factorización LU se simplifica de forma notable dando lugar a la llamada factorización de Choleski, donde $A = LU = LL^T$ obteniéndose un considerable ahorro en uso de memoria y CPU.

En muchos casos, la resolución de los sistemas lineales involucra la resolución de las llamadas matrices "sparse" o huecas. Estas matrices tienen un gran número de estradas nulas. También ocurre que los sistemas que aparecen el MEF, sobre todo en 3D, son de un orden muy elevado.

$$A = \begin{bmatrix} a_{11} & 0 & a_{13} & 0 & 0 & 0 \\ 0 & a_{22} & 0 & 0 & 0 & a_{26} \\ a_{31} & 0 & a_{33} & a_{34} & a_{35} & 0 \\ 0 & 0 & a_{43} & a_{44} & 0 & a_{46} \\ 0 & 0 & a_{53} & 0 & a_{55} & 0 \\ 0 & a_{62} & 0 & a_{64} & 0 & a_{66} \end{bmatrix}$$

Tenemos, pues, dos condicionantes: por un lado **las matrices son huecas y además de un orden muy elevado.**

Claramente debemos aprovechar estas propiedades de las matrices a la hora de su resolución.

Muchos han sido les esfuerzos empleados para la resolución de matrices huecas de orden elevado mediante los métodos directos: técnicas de almacenamiento que aprovechan el alto número de entradas nulas de la matriz, como es el almacenamiento en banda, en "línea de cielo" (skyline), etc.

También las técnicas de reordenamiento para minimizar el espacio de almacenamiento y también el llamado "efecto de relleno" (fill-in) que se produce en la resolución por métodos directos. (ver Almeida[2]).

Sin embargo, el orden de los sistemas introduce un nuevo factor que hace inestables a los métodos directos de resolución.

La aparición de los computadores, su elevada velocidad de cálculo y su automatismo, hicieron posibles la resolución de grandes sistemas y dieron un nuevo auge a los llamados métodos iterativos de resolución.

MÉTODOS ITERATIVOS

Estos métodos, por contraposición a los directos, se basan en la obtención de una secuencia de soluciones aproximadas $\{\mathbf{x}_j\}_{j=1,\ldots,n}$ a partir de una solución inicial $\mathbf{x}_0$ mediante un esquema del tipo $\mathbf{x}_{j+1} = P(\mathbf{x}_j,\ldots,\mathbf{x}_0)$. La anterior secuencia debe ser convergente de forma que $\lim_{n\to\infty} \mathbf{x}_n = \mathbf{x}$ donde "$\mathbf{x}$" es la solución exacta del sistema.

Los métodos iterativos vienen afectados del llamado error de truncamiento ya que, obviamente, nunca se podrá realizar de forma efectiva el paso al límite. Sin embargo, los errores de redondeo no se transmiten de forma tan acusada como en los métodos directos en las diversas fases de su implementación; lo que los hace apropiados para la resolución de sistemas de orden elevado.

Dentro de los métodos iterativos, podemos considerar los métodos básicos (Jacobi, Gauss-Seidel, SOR, SSOR, Chebyshev, etc.) y los métodos avanzados entre los que se encuentran los llamados métodos de Krylov (Gradiente Conjugado, Doble Gradiente Conjugado, GMRES, CGR, QMR, etc.)

Métodos Iterativos Básicos

Estos métodos se pueden emplear como tales, o bien, como auxiliares o "precondicionadores" de métodos más avanzados (precondicionador SSOR, Método Multimalla, etc.)

Los métodos iterativos básicos se fundamentan en una descomposición aditiva de la matriz del sistema. Dado $\mathbf{Ax} = \mathbf{b}$, el proceso de actualización viene dado por $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \mathbf{M}^{-1}\left(\mathbf{b} - \mathbf{Ax}^{(k)}\right)$; claramente la elección de "M" es crucial para asegurar la convergencia de estos métodos.

La anterior formulación se puede escribir $\mathbf{Mx}^{(k+1)} = \mathbf{Nx}^{(k)} + \mathbf{b}$ donde $\mathbf{A} = \mathbf{M} - \mathbf{N}$

Si consideramos la descomposición aditiva de la matriz en forma de $\mathbf{A}=\mathbf{D}+\mathbf{L}+\mathbf{U}$ donde "D" es la diagonal principal siendo "U" y "L" las partes estrictamente triangulares superior e inferior, respectivamente, de la matriz "A", podemos obtener:

Método de Gauss-Jacobi: $\mathbf{M}=\mathbf{D};\ \mathbf{N}=-(\mathbf{L}+\mathbf{U})$. La convergencia de este método depende de los valores propios de $\mathbf{M}^{-1}\mathbf{N}$, en concreto, será convergente cuando $\rho(\mathbf{M}^{-1}\mathbf{N})<1$

Cuando $\mathbf{M}=\mathbf{D}+\mathbf{L};\ \mathbf{N}=-\mathbf{U}$ obtenemos el llamado Método de Gauss-Seidel. Este método es una mejora sobre el anterior en el sentido de que usa mejor la información actualizada.

Si introducimos un parámetro $\omega\in\Re$ y consideramos la siguiente fórmula de iteración: $\mathbf{M}_\omega\mathbf{x}^{(k+1)}=\mathbf{N}_\omega\mathbf{x}^{(k)}+\omega\mathbf{b}$; donde las matrices se definen como: $\mathbf{M}_\omega=\mathbf{D}+\omega\mathbf{L};\ \mathbf{N}_\omega=(1-\omega)\mathbf{D}-\omega\mathbf{U}$, se obtiene el Método de Relajación Sucesiva o SOR. Cuando "ω" es menor que 1, obtenemos la subrelajación y cuando es mayor la sobrerrelajación. Si $\omega=1$, obtenemos el método de Gauss-Seidel. El método SOR se puede considerar como un Gauss-Seidel acelerado.

Otro método de aceleración muy importante desde el punto de vista teórico es el método de Chebyshev, basado en la aproximación polinómica del mismo nombre. Este método nos proporciona una cota de la velocidad de convergencia de los métodos iterativos, mediante el siguiente resultado teórico (ver Golub, Van Loan[24]):

Si $\mathbf{M}^{-1}\mathbf{A}$ es simétrica y definida positiva y aplicamos el Método de Chebyshev obtenemos la acotación:

$$\left\|\mathbf{y}^{(k)}-\mathbf{x}\right\|_2 \le 2\left(\frac{\sqrt{\kappa_2\left(\mathbf{M}^{-1}\mathbf{A}\right)}-1}{\sqrt{\kappa_2\left(\mathbf{M}^{-1}\mathbf{A}\right)}+1}\right)^k \left\|\mathbf{x}^{(0)}-\mathbf{x}\right\|_2$$

Vector de inicio y criterio de parada

Normalmente, y si no hay información adicional, por simplicidad computacional, se suele elegir como vector de inicio el vector nulo.

En lo referente a los criterios de parada, podemos considerar varias posibilidades:

1. $\left\|\mathbf{b}-\mathbf{A}\mathbf{x}^{(k)}\right\|_2 \le \varepsilon$
2. $\dfrac{\left\|\mathbf{b}-\mathbf{A}\mathbf{x}^{(k)}\right\|_2}{\left\|\mathbf{b}-\mathbf{A}\mathbf{x}^{(0)}\right\|_2} \le \varepsilon$
3. $\dfrac{\left\|\mathbf{b}-\mathbf{A}\mathbf{x}^{(k)}\right\|_2}{\left\|\mathbf{b}\right\|_2} \le \varepsilon$ que es equivalente al anterior cuando el vector de inicio es nulo.
4. $\dfrac{\left\|\mathbf{b}-\mathbf{A}\mathbf{x}^{(k)}\right\|_2}{\left\|\mathbf{x}^{(k)}\right\|_2} \le \dfrac{\varepsilon}{\left\|\mathbf{A}^{-1}\right\|_2}$

Quizá de todos ellos el mejor sea el criterio (3) ya que se tiene la siguiente acotación en función del número de condicionamiento de A : $\kappa_2(\mathbf{A})$

$$\frac{\left\|\mathbf{x}-\mathbf{x}^{(k)}\right\|_2}{\left\|\mathbf{x}\right\|_2} \le \kappa_2(\mathbf{A})\frac{\left\|\mathbf{b}-\mathbf{A}\mathbf{x}^{(k)}\right\|_2}{\left\|\mathbf{b}\right\|_2} \le \varepsilon$$

Además se tiene el adecuado efecto de "escalamiento" en la evaluación de este criterio de parada.

En algunas ocasiones se han de tener en cuenta criterios de orden físico, dependiendo del problema que se intente resolver.

En cualquier caso se ha de tener siempre en cuenta la precisión de la máquina *"u"* de manera que la norma del residuo nunca podrá ser menor que $\kappa(\mathbf{A})u$.

Esto impone restricciones en la tolerancia impuesta a la solución final que dependen, no sólo de la precisión de la máquina empleada, sino también del condicionamiento del sistema.

Métodos de Krylov

En los Métodos Básicos hemos podido obtener las sucesivas actualizaciones mediante fórmulas recursivas, lo que hace que:

$$\mathbf{x}_i \in \mathbf{x}_0 + L\left\{\mathbf{M}^{-1}\mathbf{r}_0, \mathbf{M}^{-1}\mathbf{A}\left(\mathbf{M}^{-1}\mathbf{r}_0\right), \ldots, \left(\mathbf{M}^{-1}\mathbf{A}\right)^{i-1}\left(\mathbf{M}^{-1}\mathbf{r}_0\right)\right\}$$

$$K^i(\mathbf{M}^{-1}\mathbf{A}; \mathbf{M}^{-1}\mathbf{r}_0) = L\left\{\mathbf{M}^{-1}\mathbf{r}_0, \mathbf{M}^{-1}\mathbf{A}\left(\mathbf{M}^{-1}\mathbf{r}_0\right), \ldots, \left(\mathbf{M}^{-1}\mathbf{A}\right)^{i-1}\left(\mathbf{M}^{-1}\mathbf{r}_0\right)\right\}$$

Siendo éste el llamado Subespacio de Krylov.

El primer método que usa explícitamente el Subespacio de Krylov es el Método de Gradiente Conjugado (CG) que se puede aplicar cuando la matriz es simétrica y definida positiva.

Este método nos da la siguiente acotación en las sucesivas iteraciones:

$$\left\|\mathbf{x}^{(k)} - \mathbf{x}\right\|_A \leq 2\left(\frac{\sqrt{K_2(\mathbf{A})} - 1}{\sqrt{K_2(\mathbf{A})} + 1}\right)^k \left\|\mathbf{x}^{(0)} - \mathbf{x}\right\|_A$$

Cuando la matriz no es simétrica o positiva, tenemos varios métodos de resolución como son el Bi-CGStab o el GMRES.

Métodos Iterativos en Espacios de Krylov generalizados

Cuando pasamos a matrices generales no simétricas y/o no positivas las dos propiedades importantes que caracterizan al método de gradiente conjugado: "minimalidad" y "optimalidad" (mínima longitud de recurrencia y óptima minimización del residuo); no se pueden seguir manteniendo simultáneamente en la implementación de los métodos de resolución.

Los métodos iterativos generalizados basados en el subespacio de Krylov forman una familia de algoritmos de mucho interés en la resolución iterativa de sistemas de ecuaciones lineales para matrices generales no singulares, por tanto, de aplicación general al ámbito de sistemas con matriz no simétrica tanto reales como complejas.

Dado un vector inicial x_0, que aproxima la solución del sistema A x = b, y el vector residual r_0, $r_0 = b - A x_0$, se define el espacio de Krylov para la matriz A y el vector r_0:

$$K^i(A;r_0) = L\{r_0, Ar_0, A^2r_0, \ldots, A^{i-1}r_0\}$$

Esencialmente existen tres vías para abordar la resolución en el caso no simétrico:

1) A través de la llamada "Ecuación Normal" : $A^TAx = A^Tb$ con métodos de tipo gradiente.

2) Mediante la construcción de una base del Subespacio de Krylov considerando relaciones biortogonales de tres términos.

3) Determinando explicitamente todos los residuos para la obtencion de una base ortonormal del subespacio de Krylov (Arnoldi, Gramm-Schmidt), (ver Brown[7])

Usando la primera opción surgen los llamados métodos de gradiente conjugado de la normal: CGN tambien conocido como CGNR o CGNE y también destaca el llamado LSQR (mínimos cuadrados) que también se puede aplicar en la resolución de sistemas sobredeterminados.

Entre ellos se tiene el método CGN en el que a partir de la ecuación:

$$x_n = x_0 + \left\langle A^T r_0, (A^T A) A^T r_0, (A^T A)^2 A^T r_0, \ldots, (A^T A)^{n-1} A^T r_0 \right\rangle$$

y minimizando $\|r_n\|_2$ en el subespacio de Krylov se obtiene que:

$$r_n \perp L\left\{(A^T A) r_0, (A^T A)^2 r_0, \ldots, (A^T A)^n r_0\right\}$$

El resultado de convergencia que se obtiene en este procedimiento es:

$$\frac{\|r_n\|}{\|r_0\|} \leq 2\left(\frac{\kappa - 1}{\kappa + 1}\right)^n$$

También se demuestra que, en aritmética exacta, el método converge a la solución "exacta" en "n" iteraciones, al igual que el gradiente conjugado.

El inconveniente del anterior método viene dado por la posible propagación, a través del cálculo de los sucesivos residuos, de los errorres de redondeo; que puede producir un estancamiento en la tasa de convergencia en las aplicaciones reales, que le hacen poco útil en diversas ocasiones.

Por ello se puede elegir la alternativa de recurrir al método LSQR, que es más robusto frente a la propagación de errorres de redondeo.

Este método se basa en la aplicación del método de Lanczos al sistema auxiliar:

$$\begin{pmatrix} I & A \\ A^T & 0 \end{pmatrix}\begin{pmatrix} r \\ x \end{pmatrix}=\begin{pmatrix} b \\ 0 \end{pmatrix}$$

Entre los métodos que se pueden encuadrar en el apartado 2), podemos considerar: el BiCG, QMR, CGS y más recientemente el BiCGStab. Estos métodos tienen cortas recurrencias (minimalidad), pero no la propiedad de optimalidad, además exige que la matriz del sistema sea positiva.

El método BiCG se basa en la aplicación del gradiente conjugado al sistema auxiliar:

$$\begin{pmatrix} 0 & A \\ A^T & 0 \end{pmatrix}\begin{pmatrix} r \\ x \end{pmatrix}=\begin{pmatrix} b \\ b \end{pmatrix}$$

El inconveniente del anterior método es la necesidad de obtener explícitamente A^T (como en todos los métodos anteriores), sumado a un mayor coste computacional.

Variantes de este método como el CGS consiguen disminuir el coste computacional al no requerir multiplicaciones por la transpuesta de "A", no obstante presenta en muchas ocasiones un comportamiento de convergencia irregular, siendo, a veces, divergente; de hecho el CGS "converge" o "**diverge**" más rápidamente respecto al BiCG en un factor entre 1 y 2.

El QMR basado en la descomposición triangular QR, presenta un comportamiento de convergencia más regular que el BCG pero no mejora de forma significativa la velocidad de convergencia.

El método BiCGStab se basa el expresar los residuos sucesivos de la siguiente forma: $r_j = Q_j(A)P_j(A)r_0$ donde $P_j(x)$ y $Q_j(x)$ son polinomios y particularmente,

$$Q_j(x) = (1-\omega_1 x)(1-\omega_2 x)\ldots(1-\omega_j x)$$

Donde el valor de ω_j en la correspondiente iteración se elige de forma que minimice el residuo.

Exige, en comparación con el CGS, la obtención de dos productos escalares adicionales en cada iteración, presenta la ventaja de ser de corta recurrencia y "cuasi-optimalidad", no hay, de momento, resultados teóricos de convergencia aunque los experimentos numéricos parecen indicar un comportamiento suave de convergencia y mucho menos errático que el CGS. Existen diversas variantes de este método según el procedimiento elegido para la minimización de los ω_j.

La tercera vía nos conduce a los métodos de residuo mínimo generalizado, en los que las recurrencias son largas, pero se mantiene la deseada optimalidad. Ello produce métodos robustos y fiables de aplicación general (GMRES), que además no exigen la positividad de la matriz del sistema.

Para una discusión y comparación entre los diversos métodos, así como los detalles de implementación computacional, y otras variantes de los mismos, se sugiere la consulta de: Nachtigal[33], Sonneveld[46], Van der Vorst[49], [50], Gutknecht[25], Vuik[51], Paige y Saunders[33].

AVANCES EN EL MÉTODO DE RESOLUCIÓN

PROPUESTA DE RESOLUCIÓN DEL SISTEMA DE ECUACIONES

Saad y Schultz propusieron en 1986 el método de Residuo mínimo generalizado: "Generalized Minimum Residual" (**GMRES**) para la resolución numérica de un sistema lineal no singular de ecuaciones algebraicas:

$$\mathbf{Ax} = \mathbf{b} \qquad (1)$$

El **GMRES** es capaz de resolver este sistema incluso sin requerirse condiciones restrictivas, como pudiera ser el exigir que la matriz **A** sea una matriz simétrica y/o positiva.

Así disponemos de una de las más serias alternativas de método de resolución general de un sistema de ecuaciones de orden alto, por su robustez y las posibilidades de vectorización y paralelización del método.

Sin embargo, el método GMRES puede consumir mucho tiempo de CPU y/o memoria. De hecho, hay cierta clase de matrices (matrices circulantes) en las cuales el GMRES sólo resuelve exactamente después de invertir un número de iteraciones igual al orden del sistema.

Hay otros problemas en la implementación práctica del GMRES completo o estándar: la norma del residuo no se calcula directamente como $\|\mathbf{b} - \mathbf{Ax}\|$ sino que se evalúa en forma indirecta como $\|\beta \mathbf{e}_1 - \mathbf{H}_k \mathbf{y}_k\|$; esto conduce a errores de redondeo que pueden desvirtuar la convergencia del método.

Aparte de la formulación clásica del GMRES, podemos considerar su encuadre, desde el punto de vista teórico, en una clase más extensa de los algoritmos ABS (Abaffi, Broyden, Spedicato); cuya formulación sería la siguiente:

Formulación ABS del método GMRES

La clase ABS de algoritmos, (ver Abaffy y Spedicato[1]), es la clase más general de algoritmos de la forma

$$x_{i+1} = x_i - a_i p \qquad (1.1)$$

Con la particularidad de que cuando la aplicamos para resolver el sistema lineal

$$Ax = b, \quad x \in R^n, \quad b \in R^m, \qquad (1.2)$$

Donde $A = (a_1,...,a_m)^T$, la solución x^* se obtiene algún punto inicial arbitrario x_1 en como máximo m pasos, $x^* = x_{m+1}$.

La clase se basa en el siguiente procedimiento.

El algoritmo ABS escalado

Sea $x_1 \in R^n$ arbitrario, sea $H_1 \in R^{nxn}$ arbitrario no singular, asignamos $i = 1$ y $iflag = 0$.

Sea $v_i \in R^m$ arbitrario tal que $v_1,...,v_i$ sean linealmente independientes. Calculamos el vector residual r_i. Si $r_i = 0$, parar, x_i son las soluciones del sistema. Si no es así, calculamos el escalar $\tau_i = v_i^T r_i$ y el vector $s_i = H_i A^T v_i$.

Si $s_i \neq 0$, ir a (4); si $s_i = 0$ y $\tau = 0$, igualamos $x_{i+1} = x_i$, $H_{i+1} = H_i$, *i*flag = *i*flag +1 e ir a (7) si $i < m$; en caso contrario parar; si $s_i = 0$ y $\tau \neq 0$, igualamos *i*flag = $-i$ y paramos.

Calculamos el vector de búsqueda p_i mediante $p_i = H_i^T z_i$ donde $z_i \in R^n$ es un vector arbitrario que cumple la condición $z_i^T H_i A^T v_i \neq 0$.

Actualizar la aproximación de la solución mediante $x_{i+1} = x_i - \alpha_i p_i$, donde α_i se obtiene mediante $\alpha_i = \tau_i / v_i^T A p_i$. Si $i = m$, parar. x_{m+1} es la solución del sistema.

Actualizar la matriz H_i mediante $H_{i+1} = H_i - H_i A^T v_i w_i^T H_i$, donde $w_i \in R^n$ es una vector arbitrario que cumple la condición $w_i^T H_i A^T v_i = 1$

Incrementar el índice i en una unidad y volver a (2).

El vector v_i es llamado vector escalante y la elección $v_i = Ap_i$, la cual está bien definida si A tiene rango máximo por columnas, define la también llamada subclase ortogonalmente escalada, que tiene la siguientes propiedades (ver Bodon[4],[5],[6]):

- Los vectores escalantes v_i son ortogonales
- Los vectores de búsqueda p_i son A^T A-conjugados
- El vector x_{i+1} minimiza la norma Euclídea del residuo en la subvariedad lineal $x_1 + L\{p_1,...,p_i\}$.

La subclase ortogonalmente escalada contiene dos algoritmos, uno correspondiente a las elecciones de parámetros $z_i = w_i = e_i$, que es la versión ABS del algoritmo QR, siendo la factorización implícita asociada del tipo QR, y otro algoritmo, correspondiente a la elecciones $z_i = w_i = A^T r_i$, que es la versión del algoritmo del gradiente conjugado de residuo mínimo.

El método GMRES, debido a Saad y Schultz (1986), es también un miembro de la subclase ortogonalmente escalada, en el sentido que la secuencia x_i producida por este método coincide con la secuencia producida por el método en esta subclase. La forma de producir esta secuencia es obviamente distinta, aparte del hecho de que, en la práctica, el método GMRES se usa como un proceso iterativo.

Este resultado es por supuesto predecible en vista de la generalidad de la formulación ABS (esencialmente sólo aquellos algoritmos que requieren más de n iteraciones para resolver un determinado sistema lineal o que fallen desde algún punto inicial, no pertenecerán a la clase ABS).

EL ALGORITMO GMRES

El método GMRES se introdujo para resolver sistemas lineales no singulares. Una formulación standard del método es la siguiente (ver Saad [42]):

(A) Elegimos un x_1 arbitrario y calcular $r_1 = b - Ax_1$ y $v_i = r_1 / \|r_1\|$. Sea $k = 1$.

(B) Para j = 1,...,k , calculamos vía el proceso de ortonormalización de Arnoldi (ver Brown[7])

$$h_{ij} = (Av_j, v_i), \quad i = 1,...,j$$

$$\hat{v}_{j+1} = Av_i - \sum_{i=1}^{j} h_i j v_i$$

Si $\hat{v}_{j+1} = 0$, parar. x_k es la solución del sistema. Si no, calculamos

$$h_{j+1,j} = \| \hat{v}_{j+1} \| \qquad v_{j+1} = \frac{\hat{v}_{j+1}}{h_{j+1,j}}.$$

(C) Teniendo en cuenta que $\mathbf{r}^{(n+1)} = \mathbf{V}_{k+1}\left[\beta^{(n)} e_1 - \mathbf{H}_k \mathbf{y}_k\right]$ (ver Saad[42]), siendo V_{k+1} ortonormal, resolver el problema lineal de mínimos cuadrados:

$$\min_{y \in R^k} \left\| \beta e_1 - \hat{H}_k y \right\|_2$$

Donde e_1 es el primer vector unidad en R^{k+1}, $\hat{H}_k$ es una matriz (k+1) x k superior de Hessenberg cuyos elementos son los h_{ij}, y $\beta = \|r_1\|$. Sea y_k la solución del anterior problema de mínimos cuadrados. La forma de la solución aproximada es

$$x_{k+1} = x_1 + V_k y_k$$

Donde $V_k = [v_1, \ldots, v_k]$. Si k = n, parar, x_{n+1} es la solución del sistema. Si no es así, incrementar "k" a "k+1" y volver a (B).

Ahora demostraremos que el algoritmo de la subclase ortogonalmente escalada, con los siguientes parámetros:

$$H_1 = Y;\ z_1 = r_1;\ z_i = Ar_{i-1};\ y>1;\ w_i = \alpha z_i;\ (\alpha \neq 0 \text{ arbitrario})$$

Genera la misma secuencia $\{x_i\}$ que el método GMRES si $m = n$.

La prueba de ello consiste simplemente en demostrar que:

$$L\{p_1,...,p_i\} = L\{r_1, Ar_1,...,A^{i-1}r_1\} = S_i.$$

En efecto: ya que el x_{i+1} obtenido por el método GMRES minimiza la norma Euclídea del residuo en $x_1 + S_i$ y la misma propiedad se cumple para todos los métodos en la subclase ortogonalmente escalada; se sigue de la unicidad de tal minimizador, que los dos métodos generan el mismo x_{i+1}.

Demostraremos por inducción que en la subclase ABS ortogonalmente escalada, para $1 \le i \le n$, se cumplen las siguientes proposiciones:

i) $r_i \in L\{r_1, Ar_1,...,A^{i-1}r_1\}$

ii) $p_i \in L\{r_1, Ar_1,...,A^{i-1}r_1\}$

Para $i = 1$,i) y ii) obviamente son ciertos.

Suponemos que i) y ii) son verdaderos para i = m.

Para $i = m + 1$, desde $r_m \in L\{r_1, Ar_1,...,A^{m-1}r_1\}$, tenemos

$$\begin{aligned} r_{m+1} &= Ax_{m+1} - b = A(x_m - \alpha_m p_m) - b = \\ &= Ax_m - \alpha_m Ap_m - b = \\ &= r_m - \alpha_m Ap_m \\ &\in L\{r_1,\ldots,A^m r_1\}; \end{aligned}$$

$$\begin{aligned} p_{m+1} &= H_{m+1}^T z_{m+1} = \left(I - \sum_{j=1}^{m} \frac{p_j p_j^T A^T A}{p_j^T A^T A p_j} \right) Ar_m = \\ &= Ar_m - \sum_{j=1}^{m} \left(\frac{p_j^T A^T A A r_m}{p_j^T A^T A p_j} \right) p_j \\ &\in L\{r_1,\ldots,A^m r_1\} \end{aligned}$$

Por lo tanto, i) y ii) se cumplen para todo i

Como $p_1,...,p_i$ son linealmente independientes, obtendremos que

$$L\{p_1,...,p_i\} = L\{r_1, Ar_1,...,A^{i-1}r_1\} = S_i.$$

q.e.d.

El GMRES truncado

Hay una alternativa al GMRES completo (estándar): la posibilidad de reiniciar tras "k" iteraciones. Esto es denominado GMRES "k-truncado" o GMRES(k).

En este método se reinicia el algoritmo después de "k" iteraciones y recalculamos "directamente" los residuos.

Nosotros utilizamos una variante de éste último método que requiere mucha menos memoria.

Debemos tener en cuenta, sin embargo, las restricciones de este método.

Para mejorar la convergencia del método usamos una versión precondicionada, propuesta por Saad y denominada método FGMRES(k) (ver Saad[41]).

Usamos un nuevo enfoque a uno de sus pasos esenciales: la resolución del problema (P):

"Encontrar $\mathbf{y}_k \in R^k$ solución del problema:

$$argmin_y \left\| \mathbf{H}_k \mathbf{y} - \beta^{(n)} \mathbf{e}_1 \right\|_2 ".$$

Esto se hace aprovechando la configuración especial del problema (P), siendo $\mathbf{H}_k$ una matriz "casi" triangular y $\mathbf{e}_1$ el primer vector de la base canónica en R^{k+1}.

Así dado un conjunto $\{\mathbf{P}_i^{-1}\}_{i=1...k}$ de matrices precondicionadoras,

1.- Inicio: Elegido $\mathbf{x}^{(0)}$ y una dimensión "k" del subespacio de Krylov; evaluamos

$$\mathbf{r}^{(0)} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(0)},\ \beta^{(0)} = \left\|\mathbf{r}^{(0)}\right\|_2$$

$$\mathbf{v}_1 = \frac{\mathbf{r}^{(0)}}{\beta^{(0)}}$$

2.- Para *j=1, 2, ..., k*:

$$\mathbf{s}_j = \mathbf{P}_j^{-1}\mathbf{v}_j$$

$$\mathbf{w} = \mathbf{A}\mathbf{s}_j$$

$$h_{ij} = \langle w, v_j \rangle,\ i = 1,\ldots,j$$

$$\hat{\mathbf{w}} = \mathbf{w} - \sum_{i=1}^{j} h_{ij}\mathbf{v}_i$$

$$h_{j+1,j} = \left\|\hat{\mathbf{w}}\right\|_2$$

$$\mathbf{v}_{j+1} = \frac{1}{h_{j+1,j}}\hat{\mathbf{w}}$$

3.- Encontrar $\mathbf{y}_k$ el cual minimiza $\left\|\mathbf{H}_k\mathbf{y} - \beta^{(n)}\mathbf{e}_1\right\|_2$, siendo $\mathbf{e}_1$ el primer vector de la base canónica en R^{k+1}, $\mathbf{y} \in R^k$, $\mathbf{H}_k$ es la matriz de orden $(k+1)\times k$ cuyas entradas son los elementos h_{ij} obtenidos en el paso 2.

4.- Calcular la siguiente aproximación de la solución:

$$\mathbf{x}^{(n+1)} = \mathbf{x}^{(n)} + \mathbf{S}_k \, \mathbf{y}_k$$

Donde $\mathbf{S}_k$ es la matriz $N \times k$ cuyas columnas son $\{\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_k\}$

5.- Reiniciar: obtener $\mathbf{r}^{(n+1)} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(n+1)}$, $\beta^{(n+1)} = \left\|\mathbf{r}^{(n+1)}\right\|_2$; si la tolerancia es satisfecha entonces paramos; caso contrario se calcula $\mathbf{v}_1 = \dfrac{\mathbf{r}^{(n+1)}}{\beta^{(n+1)}}$, e incrementamos el contador del bucle y volvemos al paso 2.

Para la minimización en el paso anterior (3), realizamos una alternativa en lugar de la factorización tradicional **QR** de $\mathbf{H}_k$, donde $\mathbf{y}_k$ es la solución del sistema triangular:

$$\mathbf{R}\mathbf{y} = \mathbf{g}, \quad \mathbf{g} = \mathbf{Q}\beta^{(n)}\mathbf{e}_1 \qquad (2)$$

Siendo **R** una matriz $(k+1)\times k$ triangular cuya última fila es cero, y **Q** una matriz de orden $(k+1)\times(k+1)$ dadas por el producto de las matrices de rotación de Givens $\mathbf{G}_k, \ldots, \mathbf{G}_j, \ldots, \mathbf{G}_1$, donde $\mathbf{G}_j = \mathbf{G}(j, j+1, \theta_j)$[52]

Procedimiento alternativo propuesto

Multiplicando "formalmente" por la matriz $\mathbf{H}_k^t$, consideramos la proyección ortogonal sobre el subespacio de las soluciones, con lo que el problema (P) queda (ver Galán [19]):

$$\mathbf{H}_k^t \mathbf{H}_k \mathbf{y} = \mathbf{H}_k^t \beta^{(n)} \mathbf{e}_1 \qquad (3)$$

Así podemos tratarlo como un problema tipo mínimos cuadrados (un sistema lineal sobredeterminado "especial"). Obviamente la matriz producto no deberá ser calculada explícitamente, pues daría lugar a un coste innecesario como mostraremos a continuación; y, por otro lado tendríamos un indeseable incremento de la inestabilidad numérica. En lugar de ello, descomponemos el producto "formal" $\mathbf{H}_k^t\mathbf{H}_k$ en una suma como sugiere la estructura de la matriz $\mathbf{H}_k$.

Examinando $\mathbf{H}_k$ en detalle, vemos que es una matriz $(k+1)\times k$ con la estructura:

$$\mathbf{H}_k = \begin{pmatrix} \mathbf{u}_k^t \\ \mathbf{M}_k \\ \mathbf{0} \end{pmatrix}$$

La primera fila constituye un vector $\left(\mathbf{u}_k^t\right)$ de dimensión k y las siguientes forman una matriz triangular superior $\mathbf{M}_k$:

$$\begin{aligned} u_i &= h_{1i} \qquad i = 1,\ldots,k \\ m_{ij} &= h_{i+1,j} \quad i,j = 1,\ldots,k \end{aligned} \tag{4}$$

Basado en esto, mostramos el esquema de multiplicación de la matriz

$$\mathbf{H}_k^t\mathbf{H}_k = \begin{pmatrix} \mathbf{u}_i & \\ \mathbf{u}_j & \mathbf{M}_k^t \\ & \mathbf{0} \end{pmatrix} \begin{pmatrix} \mathbf{u}_k^t & \mathbf{u}_j \\ & \mathbf{M}_k \\ \mathbf{0} & \end{pmatrix}$$

(i, j)

podemos observar que el (i, j)-ésimo elemento de $\mathbf{H}_k^t\mathbf{H}_k$ es:

$$u_i u_j + \sum_{p=1}^{k} m_{pi}\, m_{pj}$$

Siendo u_i la i-ésima componente de $\mathbf{u}_k^t$ y m_{pi} el (p, i)-elemento de la matriz $\mathbf{M}_k$.Esto da lugar a:

$$\left(\mathbf{u}_k \otimes \mathbf{u}_k + \mathbf{M}_k^t \mathbf{M}_k\right) \mathbf{y} = \mathbf{H}_k^t \beta^{(n)} \mathbf{e}_1 \tag{5}$$

Observando el miembro derecho de la ecuación (3), tenemos:

$$\mathbf{H}_k^t \mathbf{e}_1 = \mathbf{u}_k \tag{6}$$

Así, usando las ecuaciones (5) y (6), obtenemos la siguiente formulación del problema:

$$\left(\mathbf{u}_k \otimes \mathbf{u}_k + \mathbf{M}_k^t \mathbf{M}_k\right) \mathbf{y} = \mathbf{u}_k \beta^{(n)} \tag{7}$$

Llevando el producto externo al segundo miembro y aplicando las propiedades asociativas de multiplicación de matrices, tendremos:

$$\mathbf{M}_k^t \mathbf{M}_k \, \mathbf{y} = \mathbf{u}_k \left(\beta^{(n)} - \langle \mathbf{u}_k, \mathbf{y} \rangle\right) \tag{8}$$

Introduciendo un parámetro escalar:

$$\lambda^{(n)} = \beta^{(n)} - \langle \mathbf{u}_k, \mathbf{y} \rangle \tag{9}$$

y considerando un vector auxiliar z:

$$\mathbf{y} = \lambda^{(n)} \mathbf{z} \tag{10}$$

por tanto,

$$\mathbf{M}_k^t \mathbf{M}_k \mathbf{z} = \mathbf{u}_k \tag{11}$$

Lo que constituye un "doble" sistema triangular, el cual puede ser fácilmente resuelto para z por "doble" sustitución regresiva con bajo coste, por ser $\mathbf{M}_k^t$ y $\mathbf{M}_k$ matrices triangulares superior e inferior respectivamente.

Después de obtenerse el vector $\mathbf{z}_k$, de dimensión "k", solución de (11), podemos obtener $\lambda^{(n)}$ desarrollando (9) y usando (10):

$$\lambda^{(n)} = \beta^{(n)} - \langle \mathbf{u}_k, \mathbf{y}_k \rangle = \beta^{(n)} - \lambda^{(n)} \langle \mathbf{u}_k, \mathbf{z}_k \rangle \tag{12}$$

Despejando obtenemos:

$$\lambda^{(n)} = \frac{\beta^{(n)}}{1 + \langle \mathbf{u}_k, \mathbf{z}_k \rangle} \tag{13}$$

Puede comprobarse que $\langle \mathbf{u}_k, \mathbf{z}_k \rangle \geq 0$:

$$\langle \mathbf{u}_k, \mathbf{z}_k \rangle = \langle \mathbf{M}_k^t \mathbf{M}_k \mathbf{z}_k, \mathbf{z}_k \rangle = \left(\left\| \mathbf{M}_k^t \mathbf{z}_k \right\|_2 \right)^2 \geq 0$$

Y por consiguiente el algoritmo no degenerará.

Finalmente, se calcula $\mathbf{y}_k$ usando (10).

El residuo y su norma

El nuevo cálculo del residuo se basa en el resultado (ver Saad[42]):

$$\mathbf{r}^{(n+1)} = \mathbf{V}_{k+1} \left[\beta^{(n)} e_1 - \mathbf{H}_k \mathbf{y}_k \right] \tag{14}$$

Usando la descomposición de $\mathbf{H}_k$ en $\left(\mathbf{u}_k^t \right)$ y $\mathbf{M}_k$, la primera componente del vector $\left(\mathbf{H}_k \mathbf{y}_k \right)$, de orden (k+1), es el producto escalar $\langle \mathbf{u}_k, \mathbf{y}_k \rangle$ y el resto de las componentes vienen dadas por el vector $\left(\mathbf{M}_k \mathbf{y}_k \right)$ de dimensión "k". El vector $\hat{\mathbf{r}}^{(n+1)}$ de orden (k+1) viene dado por:

$$\hat{\mathbf{r}}^{(n+1)} = \beta^{(n)} e_1 - \mathbf{H}_k \mathbf{y}_k \tag{15}$$

A partir de (12), su primera componente es $\lambda^{(n)}$ y el resto son ahora:

$$-M_k y_k = -\lambda^{(n)} M_k z_k = -\lambda^{(n)} \hat{z}_k \tag{16}$$

donde $\hat{\mathbf{z}}_k$ puede ser obtenido en el primer remonte de (11).

Así se reduce el cálculo del nuevo vector residual a:

$$\mathbf{r}^{(n+1)} = \mathbf{V}_{k+1} \hat{\mathbf{r}}^{(n+1)} \tag{17}$$

A partir de (17), y teniendo en cuenta que $\mathbf{V}_{k+1}$ es unitaria, tenemos

$$\left\| \mathbf{r}^{(n+1)} \right\|_2 = \left\| \hat{\mathbf{r}}^{(n+1)} \right\|_2 \tag{18}$$

Lo que supone un menor coste.

Una alternativa de resolución es, por tanto, el método *FGMRES(k)* modificado que a continuación se describe.

De acuerdo a la exposición anterior, el algoritmo modificado de FGMRES(k) es:

1.- Inicio: Elegido $\mathbf{x}^{(0)}$ y una dimensión k de los subespacios Krylov; obtenemos

$$\mathbf{r}^{(0)} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(0)}, \ \beta^{(0)} = \left\| \mathbf{r}^{(0)} \right\|_2 \text{ y } \mathbf{v}_1 = \frac{\mathbf{r}^{(0)}}{\beta^{(0)}}.$$

2.- Para $j=1, 2, \ldots, k$:

$$\mathbf{s}_j = \mathbf{P}_j^{-1} \mathbf{v}_j$$

$$\mathbf{w} = \mathbf{A}\mathbf{s}_j$$

$$h_{ij} = \langle \mathbf{w}, \mathbf{v}_i \rangle, \quad i = 1, \ldots, j$$

$$\hat{\mathbf{w}} = \mathbf{w} - \sum_{i=1}^{j} h_{ij}\mathbf{v}_i$$

$$h_{j+1,j} = \|\hat{\mathbf{w}}\|_2$$

$$\mathbf{v}_{j+1} = \frac{1}{h_{j+1,j}}\hat{\mathbf{w}}$$

3.- Resolver $\mathbf{M}_k^t\hat{\mathbf{z}} = \mathbf{u}_k$ y $\mathbf{M}_k\mathbf{z} = \hat{\mathbf{z}}_k$ por doble remonte.

Obtener $\lambda^{(n)}$, $\mathbf{y}_k$ y $\hat{\mathbf{r}}^{(n+1)}$

$$\mathbf{y}_k = \lambda^{(n)}\mathbf{z}_k$$

$$\hat{\mathbf{r}}^{(n+1)} = \begin{cases} \hat{r}_1^{(n+1)} = \lambda^{(n)} \\ \hat{r}_{i+1}^{(n+1)} = -\lambda^{(n)}\hat{z}_i, \quad i = 1,\ldots,k \end{cases}$$

Con la descomposición en $\mathbf{u}_k$ y $\mathbf{M}_k$ de $\mathbf{H}_k$ dado por (4).

4.- Calcular la siguiente aproximación de la solución:

$$\mathbf{x}^{(n+1)} = \mathbf{x}^{(n)} + \mathbf{S}_k\mathbf{y}_k$$

donde $\mathbf{S}_k$ es la matriz de orden $N \times k$ cuyas columnas son $\{\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_k\}$

5.- Reiniciar: calcular $\beta^{(n+1)} = \|\hat{\mathbf{r}}^{(n+1)}\|_2$. Si la tolerancia se satisface, fin; caso contrario obtener $\mathbf{r}^{(n+1)} = \mathbf{V}_{k+1}\hat{\mathbf{r}}^{(n+1)}$, siendo $\mathbf{V}_{k+1}$ la matriz cuyas columnas son $\{\mathbf{v}_1, \ldots, \mathbf{v}_{k+1}\}$, y, $\mathbf{v}_1 = \frac{\mathbf{r}^{(n+1)}}{\beta^{(n+1)}}$. Incrementar el contador del bucle e ir al paso 2.

Estrategias de eficiencia computacional

El proceso de cálculo del vector y_k puede ser implementado fácilmente en cualquier plataforma con bajo coste y usando subrutinas bien conocidas de resolución directa.

En caso de disponerse de un "hardware" apropiado, estas subrutinas pueden ser vectorizables y paralelizables usando métodos estándar de vectorización y paralelización para sistemas lineales triangulares. (En general, y considerando los casos en que k no es excesivamente grande, quizá, la paralelización no sería "económica" teniendo en cuenta la sobrecarga que ella misma implica).

ALGORITMO DE RESOLUCIÓN GMRES(VARIABLE)

A partir de experiencias numéricas parece que el uso del método GMRES (entendiendo por éste la búsqueda de la solución para una tolerancia preseleccionada sin truncar el algoritmo) pudiera significar la mejor alternativa; no obstante esto significa, también, un alto coste en requerimientos de memoria. El uso de GMRES(k) se apoya, por otro lado, en una dimensión "k" fija para el subespacio de Krylov.

En efecto, hay un compromiso entre incrementar el valor de "k" con el objeto de alcanzar la solución numérica sin reiniciar y, por otro lado, tomar un valor fijo para la dimensión del espacio de Krylov, el cual podría darnos la solución después de varios pasos del algoritmo truncado con su consiguiente ahorro en el uso de memoria.

Es claro que para un sistema lineal cualquiera no puede ser preestablecido el valor de "*k*" que optimice el tiempo de CPU y la memoria requerida, o incluso tal que asegure la convergencia del ***(F)GMRES(k)***. Esta elección de un valor apropiado es decisiva, no únicamente para optimizar los recursos sino para obtener un esquema convergente.

Por consiguiente sería de interés determinar por adelantado el valor de dicho parámetro, sin embargo, parece que actualmente no existen tales "predictores", o en caso de estar disponibles, el coste computacional para obtenerlos sería prohibitivo por ser un problema de igual o superior orden a la propia resolución del sistema. La única posibilidad que nos queda es construir estrategias de tipo ensayo/error.

Estas consideraciones conducen al siguiente algoritmo propuesto *(F)GMRES(variable)*, cuya implementación corresponde a:

1. En un primer paso se evalúa la norma del residuo $\left\|\hat{\mathbf{r}}^{(n+1)}\right\|_2$ a través de "evaluación indirecta", considerando una nueva "subtolerancia" que puede ser en general una función $Tol' = f(Tol)$ e incrementamos "*k*" en pasos de uno en uno, buscando el valor de "*k*" que verifique $\frac{\left\|\hat{\mathbf{r}}^{(k+1)}\right\|_2}{\|\mathbf{b}\|_2} < Tol'$.

2. Habiendo alcanzado este valor de "*k*", reiniciar el algoritmo como un GMRES(k) (truncado) estándar para alcanzar la tolerancia exigida *Tol*.

Hay varias consideraciones a tener en cuenta en las aplicaciones, que debemos señalar en esta nueva aproximación al método GMRES(k):

1- El valor de *Tol'* debe ser mayor o igual que *Tol*.

2- El orden "*k*" no se conoce de antemano, por tanto, los requerimientos de memoria no son conocidos, no obstante la memoria total empleada será más pequeña que la correspondiente al GMRES no truncado.

3- Dos criterios adicionales se establecerán para determinar el máximo orden "*k*": agotamiento de la memoria disponible y/o un límite impuesto. En este caso, alcanzado este valor de "*k*", ***(F)GMRES(k)*** deberá hacer lo que "buenamente" sea posible, (en este caso "desfavorable").

4- La "subtolerancia" *Tol'* debe ser bastante pequeña para situarse dentro del dominio de atracción en el cual el ***(F)GMRES(k)*** converge.

5- Reiniciar el GMRES da la oportunidad de aprovechar sus buenas propiedades de convergencia y minimizar, de paso, el uso de memoria.

6- El GMRES truncado puede ser no mucho más costoso en tiempo de uso de CPU que el GMRES no truncado y en algunos casos puede resultar, incluso, mas económico (evitando el efecto acumulativo de los errores de redondeo).

7- Esta implementación depende, fundamentalmente, de la posibilidad de asignación dinámica de memoria.

Después de estas consideraciones, se ha construido una aproximación con el siguiente criterio:

Referente al punto primero y cuarto, y considerando que *Tol* se elige normalmente en un rango entre $10^{-5} - 10^{-10}$, se propone la elección $Tol' = \sqrt[3]{Tol}$.

En relación al último punto, la implementación se ha realizado en lenguaje ANSI-C con sus conocidas funciones de asignación dinámica de memoria, que además, a través del estándar ANSI, evita el severo inconveniente de la aritmética de doble precisión forzada de las antiguas implementaciones del lenguaje de programación. (También pudiera optarse a su realización en FORTRAN 90).

Esta implementación se ha combinado con el uso de precondicionadores tales como, el diagonal, factorización incompleta LU y Bi-CGStab (con numero de iteraciones fijo); también en este último caso, considerando la variante FGMRES(k).

El uso de GMRES como propio precondicionador propuesto por Saad[41] ha sido descartado, considerando que este uso está ligado a un esquema de asignación fija de recursos de memoria.

La implementación en ANSI-C

Esta implementación tiene las siguientes ventajas:

1- Asignación dinámica de memoria, lo que hace posible optimizar la memoria y por tanto poder tratar problemas de mayor orden.

2- Uso de direccionamiento de punteros, siendo capaces, de este modo, de hacer que las funciones devuelvan datos múltiples.

3- Alto nivel de portabilidad, a través del estándar ANSI.

Por otro lado, el código se ha implementado en el sentido de hacerle lo más próximo posible al algoritmo real, evitando el uso de estructuras complejas que obscurecerían el seguimiento de la implementación.

Algunas observaciones deben considerarse en este esquema: una de ellas es que debido a errores de redondeo, el residuo evaluado como $\|\beta\, \mathbf{e}_1 - \mathbf{H}_k \mathbf{y}_k\|$ puede diferir del que se obtendría directamente a través de $\|\mathbf{r}^{(n+1)}\| = \|\mathbf{b} - \mathbf{A}\mathbf{x}^{(n+1)}\|$.

Por eso es conveniente evaluar el residuo de esta ultima manera y por ello reiniciarlo después de haber fijado el valor de "k" (dimensión del subespacio de Krylov). Otra consideración es que el criterio de parada ha sido seleccionado de forma invariante a la escala:

$$\frac{\|\mathbf{r}^{(n+1)}\|}{\|\mathbf{b}\|} < Tol$$

PRECONDICIONADORES

GMRES Y PRECONDICIONADORES

La técnica del precondicionamiento es de uso común en los métodos iterativos de resolución de sistemas.

El fin que se persigue es el de aumentar la velocidad de convergencia del método y mejorar sus propiedades de estabilidad numérica.

Dado el sistema $\mathbf{Ax} = \mathbf{b}$, se toma una matriz $\tilde{\mathbf{A}} \cong \mathbf{A}^{-1}$ que aproxime a la matriz inversa y se resuelve el sistema equivalente $\tilde{\mathbf{A}}\mathbf{Ax} = \tilde{\mathbf{A}}\mathbf{b}$ (precondicionamiento por la izquierda o externo), o bien, $\mathbf{A}\tilde{\mathbf{A}}\tilde{\mathbf{A}}^{-1}\mathbf{x} = \mathbf{b}$ (precondicionamiento por la derecha o interno). En el caso de que la matriz $\tilde{\mathbf{A}}$ sea factorizable se puede dar también el precondicionamiento mixto o interno/externo.

TIPOS DE PRECONDICIONADORES

Se pueden considerar dos grandes categorías de precondicionadores: los precondicionadores de tipo local y los precondicionadores de tipo global. Los primeros corrigen problemas de mal condicionamiento local de la matriz y van estrechamente ligados al tipo de problema; mientras que los segundos son de tipo más general y no dependen tanto del tipo de matriz, exigiendo de ésta, únicamente, el que se satisfagan unas condiciones mínimas (entradas diagonales no nulas, etc.)

Han sido implementados varios precondicionadores en el algoritmo **(F)GMRES(v)** para mejorar/acelerar la convergencia.

Entre ellos tenemos dos categorías: precondicionadores de tipo directo (constante) y de tipo iterativo (variables).

En la primera categoría ha sido incluido el DIAGONAL, Factorización Incompleta LU Segura: SILU(0) y el SSOR.

En la segunda clase hemos considerado el BI-CGstab.

Precondicionador Diagonal

Es el más simple de los precondicionadores, implica únicamente un escalado en la matriz: dada D, diagonal principal de la matriz del sistema, se considera la resolución de $\mathbf{D}^{-1}\mathbf{A}\mathbf{x} = \mathbf{D}^{-1}\mathbf{b}$ como sistema precondicionado por la izquierda (externo) o bien $\mathbf{A}\mathbf{D}^{-1}\mathbf{D}\mathbf{x} = \mathbf{b}$ en el caso de precondicionador por la derecha (interno).

Este precondicionador tan simple puede ser en algunos casos muy efectivo ya que su efecto escalante puede eliminar peligrosos efectos de inestabilidad numérica por desescalamiento (K. Vuik. private comm.). Por otro lado, es fácilmente vectorizable/paralelizable.

Precondicionador SSOR

Este precondicionador se basa en el método de Gauss-Seidel (con parámetro de relajación) y consiste en la descomposición de la matriz en forma de suma $\mathbf{A} = \mathbf{D} - \mathbf{E} - \mathbf{F}$ donde E es el triángulo inferior de A cambiado de signo y F es el triángulo superior también cambiado de signo.

La matriz de precondicionamiento se construye como: $\tilde{\mathbf{A}} = \left[\frac{\omega}{2-\omega}\left(\frac{1}{\omega}\mathbf{D}-\mathbf{E}\right)\mathbf{D}^{-1}\left(\frac{1}{\omega}\mathbf{D}-\mathbf{F}\right)\right]^{-1}$ donde ω es el parámetro de relajación que oscila normalmente alrededor de 1.

Este precondicionador es uno de las más efectivos aunque no es tan fácilmente vectorizable/paralelizable como el diagonal al involucrar remontes de matrices triangulares.

Precondicionador ILU: SILU(0)

Este precondicionador se basa en la descomposición factorial de la matriz A=LU. Sin embargo, en nuestro caso, ésta factorización se efectuará únicamente en aquellas entradas NO NULAS de la matriz.

El principal problema que se pueden plantear en este caso el "pivote casi nulo", en este caso, se opta por poner una barrera que desactive esta posibilidad: factorización SILU.

Los dos precondicionadores anteriormente considerados, pueden ser mejorados mediante una reordenación o renumeración de la matriz del sistema.

Hay varios algoritmos propuestos para efectuar dicha renumeración, entre ellos podemos destacar el de George, Cuthill-McKee, CMK inverso, Grado Mínimo, "serpiente", etc. todos ellos de tipo determinístico (ver Almeida[2], Dutto[15], Cuthill[12], George[21], Gibbs[22], Martin[30], Tinney[48]).

También es posible una aproximación estocástica al problema mediante el uso del algoritmo de "Simulated Annealing" (ver Galán[20], Metropolis[31], Otten[36]).

Estos dos precondicionadores, por su potencia y gran efecto en la aceleración de la convergencia del método, son las elecciones más claras.

Precondicionador BI-CGStab

Este precondicionador exige la posibilidad de precondicionamiento variable del GMRES, lo que se consigue en el FGMRES.

Se basa en aplicar varias iteraciones del Bi-GCStab. Este método iterativo de resolución desarrollado en 1993 por Van der Vorst deriva del doble gradiente conjugado, aunque aprovecha una optimización polinomial para evitar el uso de matrices transpuestas.

La descripción algorítmica del método podría ser la siguiente (ver Van der Vorst[50]):

Siendo $\mathbf{x}_0$ el vector de partida

$$\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0 ; \rho_0 = \alpha = \omega_0 = 1 ; \mathbf{v}_0 = \mathbf{p}_0 = \mathbf{0}$$

Inicio de bucle: para $i = 1,2,3,\ldots$

$$\rho_i = \langle \mathbf{r}_0, \mathbf{r}_{i-1} \rangle ; \ \beta = \left(\frac{\rho_i}{\rho_{i-1}} \right) \left(\frac{\alpha}{\omega_{i-1}} \right) ; \ \mathbf{p}_i = \mathbf{r}_{i-1} + \beta (\mathbf{p}_{i-1} - \omega_{i-1} \mathbf{v}_{i-1})$$

$$\mathbf{v}_i = \mathbf{A}\mathbf{p}_i ; \ \alpha = \frac{\rho_i}{\langle \mathbf{r}_0, \mathbf{v}_i \rangle} ; \ \mathbf{s} = \mathbf{r}_{i-1} - \alpha \mathbf{v}_i ; \ \mathbf{t} = \mathbf{A}\mathbf{s}$$

$$\omega_i = \frac{\langle \mathbf{t}, \mathbf{s} \rangle}{\langle \mathbf{t}, \mathbf{t} \rangle} ; \ \mathbf{x}_i = \mathbf{x}_{i-1} + \alpha \mathbf{p}_i + \omega_i \mathbf{s} ; \ \mathbf{r}_i = \mathbf{s} - \omega_i \mathbf{t}$$

Fin de bucle

Podemos observar que no aparece "A^T".

Tablas de resultados test

Presentamos a modo de ejemplo la resolución de un sistema "sparse" de orden 16340 mediante el FGMRES clásico y nuestro FGMRES variable. Se solicita una precición de 10^{-8} (*Intel 486-DX50*)

Tabla de resultados: FGMRES.

FGMRES/SILU(0) tradicional			
Tolerancia	Orden	Trunc.	Tiempo
1	1	1	1.085
0.0001884	2	1	1.513
2.33E-05	3	1	1.98
9.72E-06	4	1	2.485
6.76E-06	5	1	3.023
4.84E-06	6	1	3.597
3.62E-06	7	1	4.207
2.97E-06	8	1	5.055
2.44E-06	9	1	5.738
2.08E-06	10	1	6.459
1.71E-06	11	1	7.215
1.52E-06	12	1	8.008
1.33E-06	13	1	8.838
1.21E-06	14	1	9.708
1.06E-06	15	1	10.81
9.52E-07	16	1	11.748
8.59E-07	17	1	12.722
7.88E-07	18	1	13.733
7.19E-07	19	1	14.781
6.55E-07	20	1	16.063
6.02E-07	21	1	17.182
5.51E-07	22	1	18.337
5.19E-07	23	1	19.528
4.81E-07	24	1	20.955
4.49E-07	25	1	22.218
4.17E-07	26	1	23.516
3.89E-07	27	1	24.852
3.67E-07	28	1	26.422
3.43E-07	29	1	27.828
3.22E-07	30	1	29.27
3.01E-07	31	1	30.75
2.84E-07	32	1	32.464
2.69E-07	33	1	34.013
2.55E-07	34	1	35.6
2.42E-07	35	1	37.421
2.28E-07	36	1	39.082
2.18E-07	37	1	40.774
2.08E-07	38	1	42.701
2.00E-07	39	1	44.463
1.92E-07	40	1	46.262
1.83E-07	41	1	48.298
1.75E-07	42	1	50.169
1.67E-07	43	1	52.075

1.60E-07	44	1	54.216
1.52E-07	45	1	56.194
1.45E-07	46	1	58.406
1.38E-07	47	1	60.455
1.31E-07	48	1	62.539
1.25E-07	49	1	64.86
1.19E-07	50	1	67.016
1.13E-07	51	1	69.408
1.08E-07	52	1	71.636
1.03E-07	53	1	74.099
9.80E-08	54	1	76.398
9.30E-08	55	1	78.731
8.84E-08	56	1	81.302
8.41E-08	57	1	83.707
8.00E-08	58	1	86.35
7.61E-08	59	1	88.826
7.25E-08	60	1	91.54
6.93E-08	61	1	94.087
6.60E-08	62	1	96.874
6.31E-08	63	1	99.492
6.02E-08	64	1	102.349
5.74E-08	65	1	105.04
5.47E-08	66	1	107.967
5.21E-08	67	1	111.04
4.97E-08	68	1	113.838
4.75E-08	69	1	116.877
4.54E-08	70	1	119.746
4.34E-08	71	1	122.854
4.10E-08	72	1	125.795
3.85E-08	73	1	128.973
3.53E-08	74	1	132.189
3.05E-08	75	1	135.236
2.39E-08	76	1	138.521
1.65E-08	77	1	141.644
1.22E-08	78	1	144.999
1.01E-08	79	1	148.393
8.68E-09	80	1	151.622

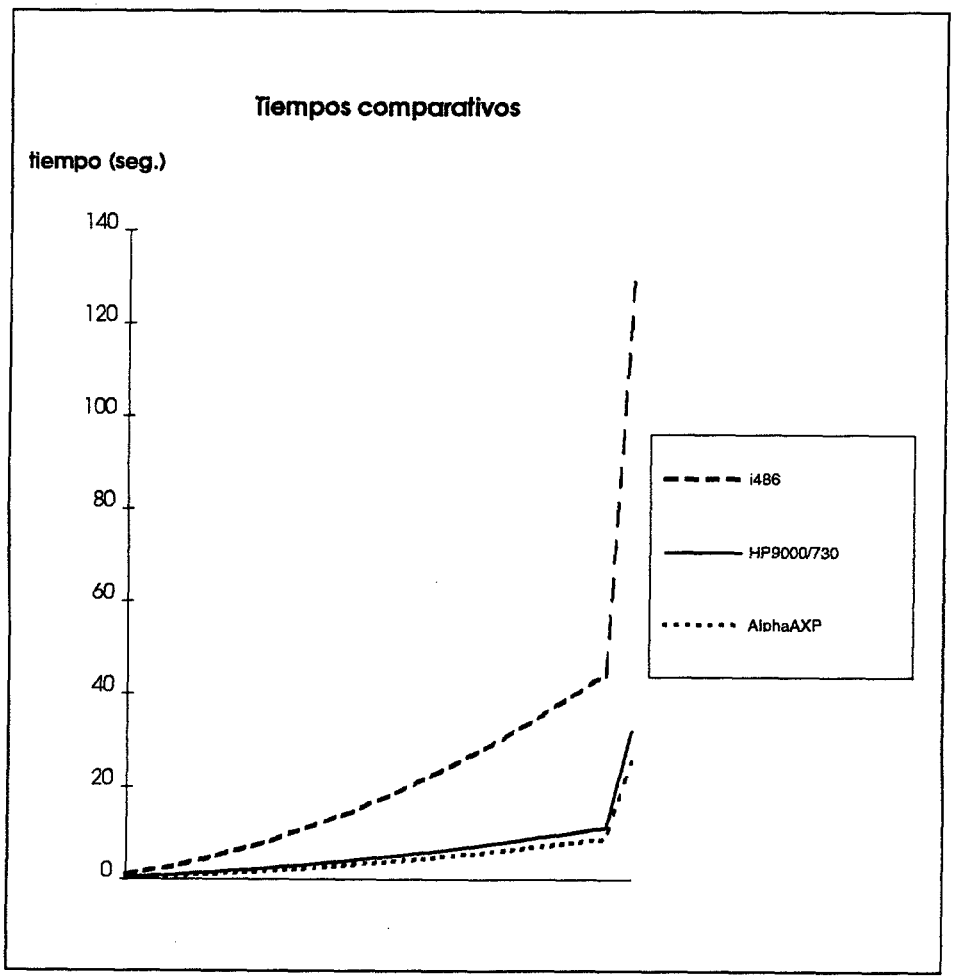
Tabla de resultados: FGMRES(variable), λ=1.2

FGMRES(var.)/SILU(0) λ=1.2			
Tolerancia	Orden	Trunc.	Tiempo
1	1	1	1.085
0.0001884	2	1	1.513
2.33E-05	3	1	1.98
9.72E-06	4	1	2.486
6.76E-06	5	1	3.027
4.84E-06	6	1	3.602
3.62E-06	7	1	4.213
2.97E-06	8	1	5.06
2.44E-06	9	1	5.745
2.08E-06	10	1	6.465
1.71E-06	11	1	7.222
1.52E-06	12	1	8.021
1.33E-06	13	1	8.851
1.21E-06	14	1	9.917
1.06E-06	15	1	10.82
9.52E-07	16	1	11.76
8.59E-07	17	1	12.737
7.88E-07	18	1	13.749
7.19E-07	19	1	14.796
6.55E-07	20	1	16.079
6.02E-07	21	1	17.199
5.51E-07	22	1	18.356
5.19E-07	23	1	19.548
4.81E-07	24	1	20.975
4.49E-07	25	1	22.238
4.17E-07	26	1	23.539
3.89E-07	27	1	24.875
3.67E-07	28	1	26.446
3.43E-07	29	1	27.854
3.22E-07	30	1	29.297
3.01E-07	31	1	30.776
2.84E-07	32	1	32.49
2.69E-07	33	1	34.042
2.55E-07	34	1	35.627
2.42E-07	35	1	37.451
2.28E-07	36	1	39.114
2.18E-07	37	1	40.807
2.08E-07	38	1	42.735
2.08E-07	38	1	43.894
3.80E-08	38	2	86.238
3.82E-09	38	3	128.621

Se comprueba mediante este ejemplo el gran ahorro de memoria que representa el uso de nustro método sobre el método clásico y tambien una cierta ganancia en los tiempos de resolución.

Tiempos Comparativos en distintas plataformas

Presentamos, a continuación, un diagrama comparativo de tiempos de resolución del anterior sistema en diferentes plataformas: AlphaAXP400 (DEC), PARISC1.1 (HP) e i486-DX50 (Intel).



ENSAMBLAJE VECTORIZACIÓN Y PARALELIZACIÓN

ENSAMBLAJE Y FORMACIÓN DE LA MATRIZ GLOBAL

El método que proponemos y que hemos considerado en la implementación informática en lenguaje C, para el ensamblaje, es el de cadenas simplemente enlazadas.

Existen varios algoritmos en lenguaje C, para tratamiento operacional de las matrices "sparse", así podemos mencionar a las cadenas enlazadas (simples y dobles), "arboles binarios", matrices de punteros, matrices "hashed", etc (ver Press[38]).

La principal cuestión en el ensamblaje de las matrices elementales es el problema de la "inserción". Así debemos insertar elementos en la matriz global, en posición adecuada, y con control de la matriz, de forma que no se destruya la estructura preexistente ni se duplique la información en memoria.

Tras efectuar el ensamblaje de las matrices elementales, proponemos almacenar la matriz ensamblada (matriz global del sistema) en forma comprimida mediante un esquema de almacenamiento hipercompacto (almacenamiento comprimido sin ninguna entrada nula).

Como se puede ver en la figura 1, hay dos estructuras definidas; la primera es de tipo diagonal y la segunda de tipo no diagonal, siendo la última "auto-referenciadora".

La matriz global es vista de este modo como un vector de estructuras diagonales, que sirven como cabecera de las listas simplemente enlazadas donde se almacenan el resto de las entradas no nulas de la matriz.

Cada uno de ellos tiene cuatro componentes:

- El valor (en doble precisión) de la entrada diagonal de la matriz.
- Un contador (entero) indicando el número de elementos no nulos en la fila.
- La cabeza de dos "cadenas simplemente enlazadas" a través de dos punteros: uno de ellos a la cadena de los elementos de la fila en la submatriz triangular superior y otra a los de la inferior.

Los elementos no diagonales se componen de tres elementos:

- El valor (en doble precisión) de esa entrada en la matriz.
- La posición de columna (entero).
- Un puntero a la estructura siguiente (de aquí la necesidad de auto-referenciamiento).

La cadena se termina con un puntero NULL del mismo tipo que la estructura.

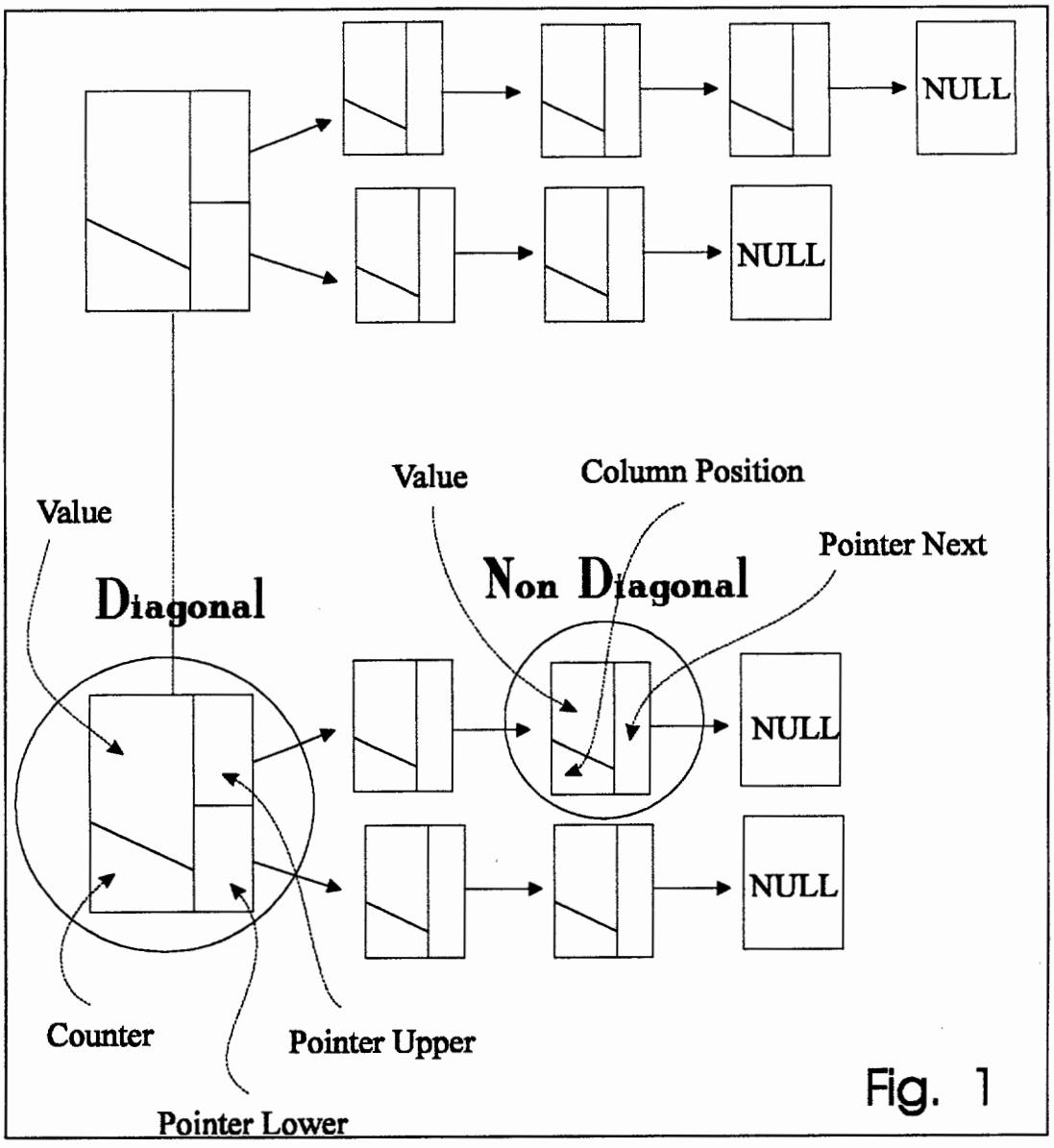


Fig. 1

Nos enfrentamos a tres posibilidades cuando insertamos las entradas de una matriz elemental:

1ª: El elemento debe ser añadido a una posición preexistente no nula: la posición de la columna coincide con uno previo (este caso siempre ocurre cuando la entrada es en la diagonal principal).

Únicamente tenemos que añadir el valor al preexistente y esto es todo, el contador de elementos no nulos permanece igual . (Fig. 2)

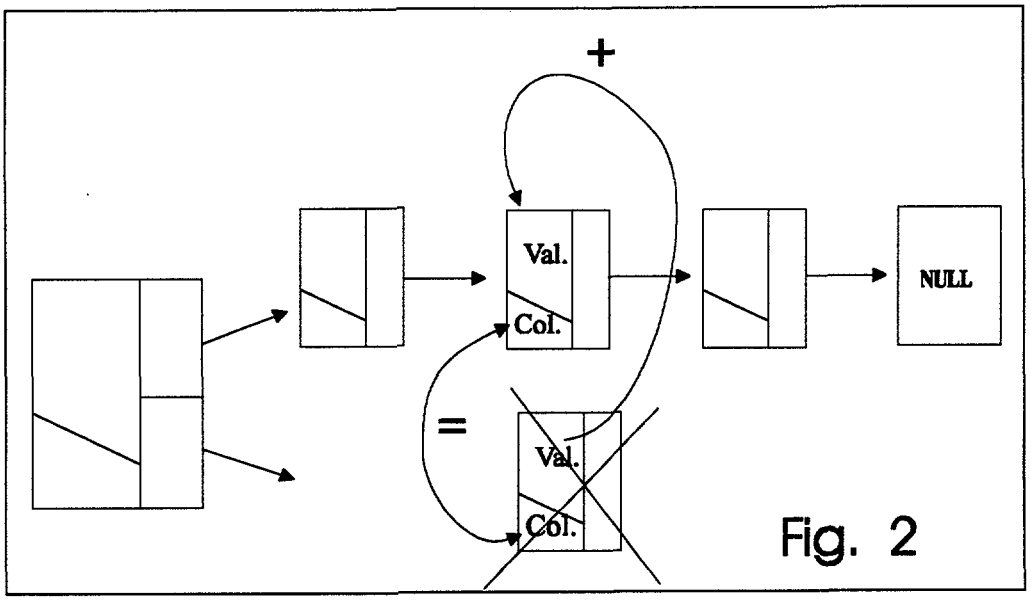


Fig. 2

2ª El elemento insertado es el primero en la lista. En este caso el contador es incrementado de 0 a 1, los punteros son redistribuidos; el esquema es resumido en la figura 3.

3º El elemento es insertado en una lista. En este caso tenemos que redistribuir los punteros e incrementar el valor del contador de "n" a "n+1".

Hay tres subclases:

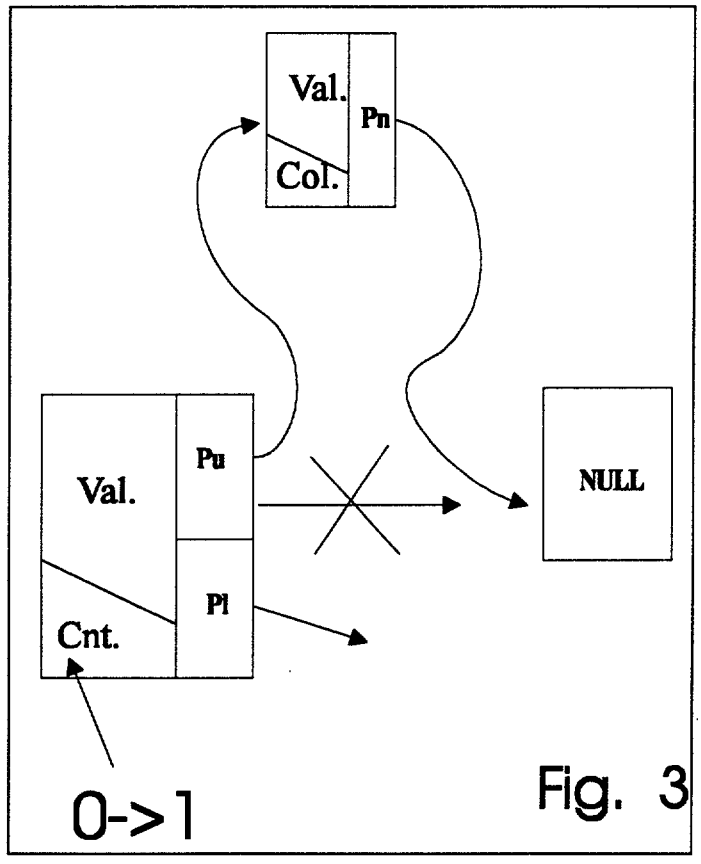
1- La inserción es en la mitad de la cadena (Fig. 4)

2.- La inserción es en la última posición (Fig. 5).

3.- La inserción ocurre en la primera posición (Fig. 6)

En cualquier caso, podemos ver que la inserción de elementos es bastante simple y directa y únicamente supone, a lo más, cuatro operaciones: búsqueda del punto de inserción, actualización de contadores y actualización de punteros.

Este procedimiento permite evitar el uso esquemas de almacenamiento de más consumo de memoria, como los esquemas de matrices en banda o de estructura de perfil simétrico.



Además, las operaciones envueltas son simples y rápidas, maximizando de este modo el beneficio del alto número de entradas nulas.

Si contamos el numero de operaciones, tenemos que el algoritmo de búsqueda en una lista simplemente enlazada es de orden $\frac{n}{2}$, siendo "n" el número de elementos en la lista; en nuestro caso "n" es muy bajo debido al alto número de entradas nulas de la matriz.

Referente a la ocupación de memoria, para una arquitectura común, tenemos que un entero ocupa 4 bytes, un puntero es 4 bytes, también y un número en doble precisión es 8 bytes, así tenemos aproximadamente Nx(16) bytes; siendo "N" el numero de elementos no nulos de la matriz.

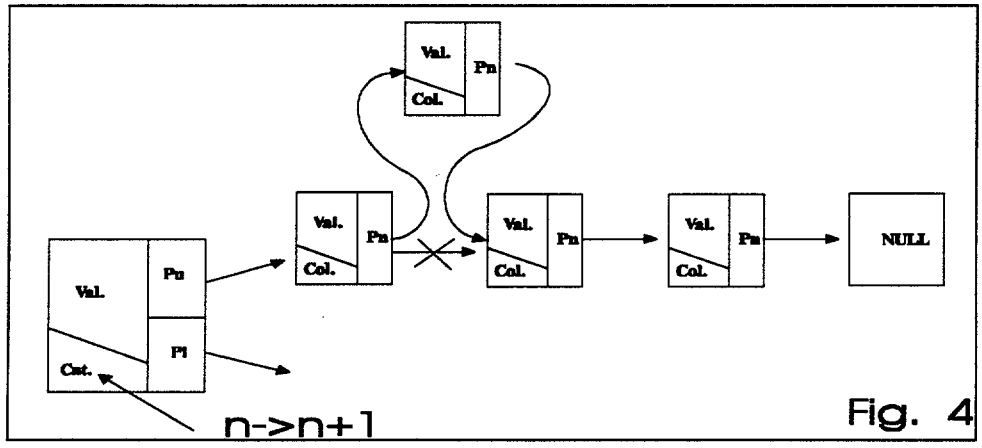


Fig. 4

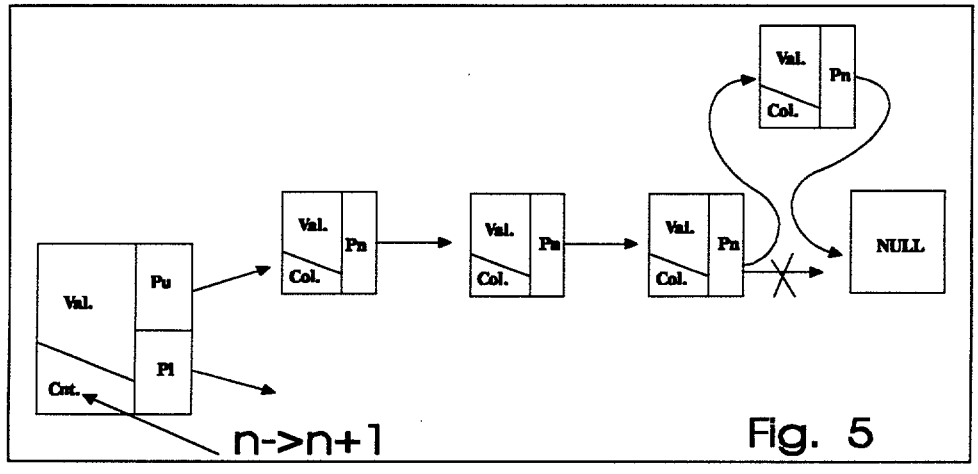


Fig. 5

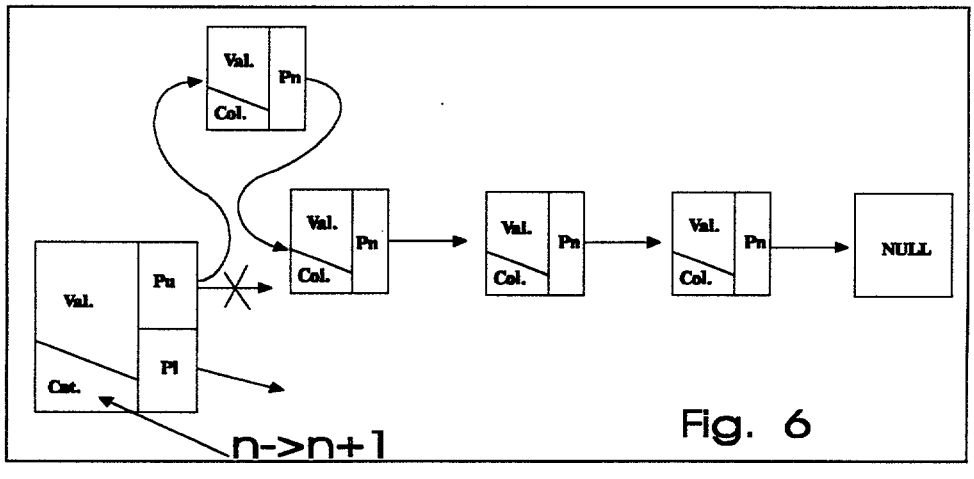


Fig. 6

ALMACENAMIENTO HIPERCOMPACTO DE LA MATRIZ DEL SISTEMA

Hemos considerado un esquema de almacenamiento o localización basado en esquema comprimido de Radicati[39], pero substancialmente modificado de acuerdo a las posibilidades que ofrece el lenguaje "C".

Así la matriz:

$$A = \begin{pmatrix} 1 & 0 & -1 & 0 & 0 & 0 & 1 \\ 0 & 2 & 0 & 1 & 2 & 3 & 0 \\ 1 & 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 4 & 1 & 0 & 0 \\ 0 & -2 & 0 & 0 & 5 & 0 & 1 \\ 0 & -3 & 0 & 0 & 0 & 6 & 0 \\ 2 & 0 & 0 & 1 & 4 & 0 & 7 \end{pmatrix}$$

Es representada por dos matrices "melladas", una de ellas contiene los valores y la otra contiene la posición de columna de esos valores del modo siguiente:

$$VA = \begin{pmatrix} 1 & -1 & 1 & \\ 2 & 1 & 2 & 3 \\ 3 & 1 & & \\ 4 & -1 & 1 & \\ 5 & -2 & 1 & \\ 6 & -3 & & \\ 7 & 2 & 1 & 4 \end{pmatrix} \qquad PA = \begin{pmatrix} 3 & 2 & 6 & \\ 4 & 3 & 4 & 5 \\ 2 & 0 & & \\ 3 & 1 & 4 & \\ 3 & 1 & 6 & \\ 2 & 1 & & \\ 4 & 0 & 3 & 4 \end{pmatrix}$$

VA es la matriz de valores en doble precisión, siendo la primera columna la diagonal y el resto los elementos no nulos.

PA es la matriz de posiciones, cada entrada en su primera columna es un índice del número de elementos no nulos en la fila.

Las otras entradas son la posición de los valores correlativos en la fila de VA (¡en lenguaje "C" el dimensionamiento comienza con índice 0!).

En este esquema, VA y PA son "vectores" de longitud variable (vectores de filas) y así mismo es su ocupación de memoria.

La implementación de este tipo de dimensionamiento de matrices es dado a través de un esquema de "doble direccionamiento" en el cual cada fila de la matriz es un vector "independiente" (Fig. 7).

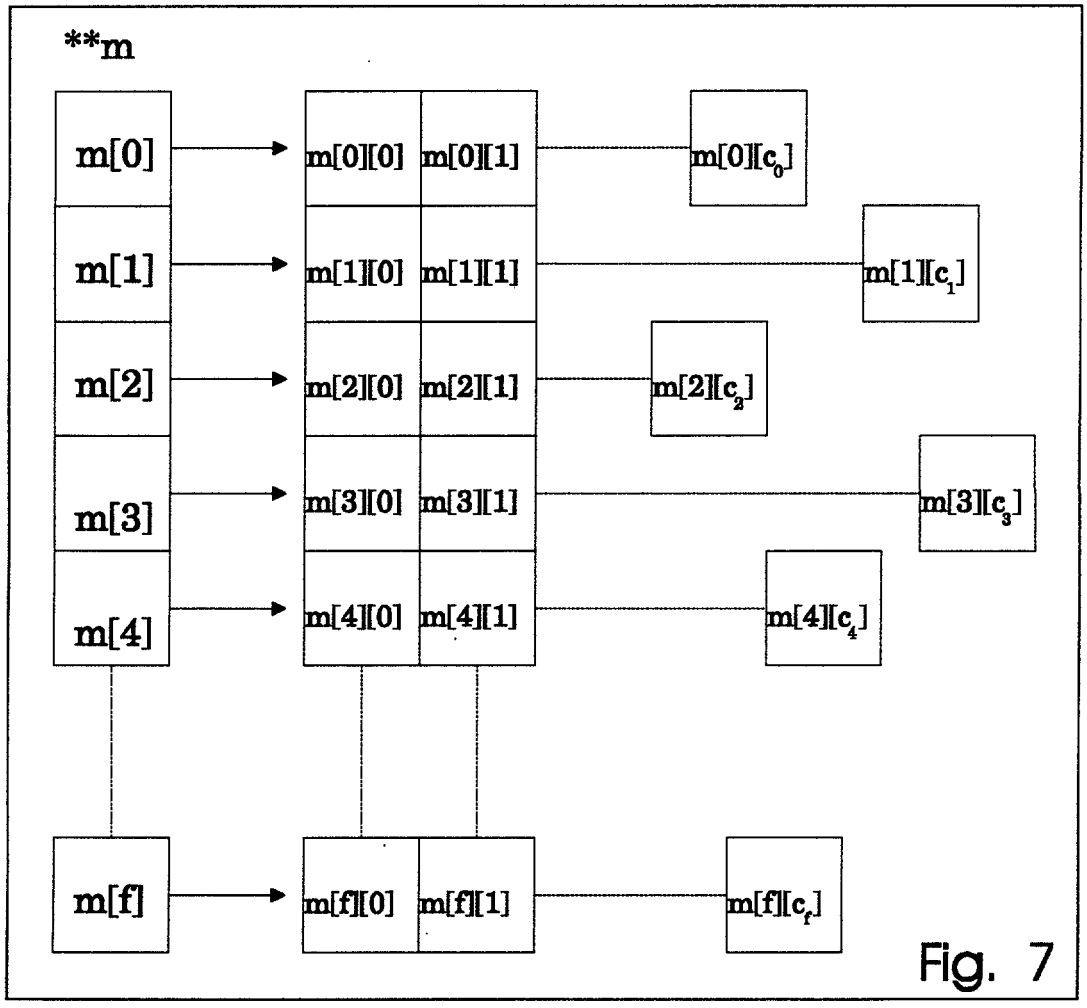


Fig. 7

También, este esquema, en contraposición al de "listas enlazadas", da la posibilidad de ejecución vectorial /paralela del producto matriz por vector.

El esquema de listas enlazadas es muy útil para el ensamblaje de la matriz global a partir de las matrices elementales en el método de elementos finitos para posterior conversión del mismo a un esquema comprimido.

CONVERSIÓN AL ALMACENAMIENTO HIPERCOMPACTO

Después del paso de ensamblaje, debemos convertir la matriz de cadenas al almacenamiento hipercompacto propuesto.

Esto debe hacerse ya que, según lo dicho anteriormente, las matrices de cadenas no son apropiadas para la optimización en el producto matriz por vector (operación de mayor coste computacional en las resoluciones iterativas) ya que este esquema de almacenamiento es "intrínsecamente" escalar (ver Dongarra[14]).

De cualquier forma, esta conversión debería ser realizada en una especie de algoritmo de tipo "reloj de arena" que rellena la matriz final mientras libera la memoria usada por la inicial.

Es preferible este esquema de almacenamiento hipercompacto frente a cualquier otra clase de almacenamiento usando direccionamiento indirecto porque tiene varias características altamente deseables: minimiza el "efecto de desplazamiento" (striding) en el acceso a la memoria; minimiza el uso de memoria y también permite la vectorización y paralelización de producto matriz por vector.

En la tabla siguiente extraída del informe de ejecución de un ordenador vectorial-paralelo TITAN 3000, podemos ver que la subrutina "xmatxvec" es ejecutada en paralelo (3 procesadores).

Podemos observar que casi todas las subrutinas que operan con vectores y matrices se equilibran bien entre los procesadores, excepto la subrutina "solvey" (la cual envuelve varias subrutinas de remonte no paralelizables).

1º	proc.	2º	proc.	3º	proc	
count	microsec	count	microsec	count	microsec	SUBR.
187	114783304	0	110561975	0	112830010	xmatxvec
1496	3268679	0	3228170	0	3242433	vecxvec
11	2734130	0	2723459	0	2735591	vecxmat
204	248178	0	240906	0	242455	vxv
187	218071	0	194380	0	210170	landav
27	32289	0	16184	0	18020	solvey
11	22800	0	22226	0	22333	upluseqv
11	22679	0	22101	0	22268	umenosv

Como vemos, la subrutina que más tiempo consume, con diferencia a las demás, es "xmatxvec" correspondiente al producto matriz por vector. Es de interés resaltar el buen comportamiento paralelo de dicha subrutina cuyos de ejecución se reparten de modo equitativo entre las tres CPUs.

Sin embargo, la subrutina "solvey", presenta un comportamiento poco satisfactorio desde el punto de vista de la paralelización: los tiempos de ejecución en cada CPU son dispares.

INFORMES DE VECTORIZACIÓN Y PARALELIZACIÓN DEL COMPILADOR

```
                    Vectorized Results From File xmatxvec.c
                           Origin -- Line 14

        Line    Stmt    Time        Program

           *       *       3        $$ds = 32;
           *       *       6        if ($$B1 < 96 & $$B1 >= 4) $$ds = $$B1>>2;
          14       *       9        DO PARALLEL ($$ip = 0; $$ip != (int) (dim) - 1; $$ip +=
$$ds) {
          14       *      15           $$rp = MIN((int) (dim) - 1, $$ip - 1 + $$ds);
           *       *       9           $$vl = $$rp - $$ip + 1;
          14       *       6           DO VECTOR ($$I1 = $$ip; $$I1 != $$rp; $$I1++) {
          16       7      48              result[$$I1] = *ac[$$I1] * v[$$I1];
                                      }
                                   }

                                Vector Parallel Statements

        Stmt Parallel Choices  Chosen      Reason  Vector Choices  Chosen      Reason

           7                1       1                              1       1        (15)

      (15)  Loop was split into two loops to get vector and parallel execution.

      Following vector ops in line 16 run at scalar speed
                           DIV (integer)
      ---------------------------------------
                    Vectorized Results From File xmatxvec.c
                           Origin -- Line 19

        Line    Stmt    Time        Program

          19       *       8        for ($$I1 = 0; $$I1 != (int) (dim) - 1; $$I1++) {
           *      21      22           $$B2 = MAX((int) (*ak[j]), 0);
           *       *       3           $$ds = 32;
           *       *       6           if ($$B2 < 96 & $$B2 >= 4) $$ds = $$B2>>2;
           *       *      20           $$LV_1 = result[j];
          21       *      31           DO PARALLEL ($$ip = 1; $$ip != (int) (*ak[j]); $$ip +=
$$ds) {
          21       *      37              $$rp = MIN((int) (*ak[j]), $$ip - 1 + $$ds);
           *       *       9              $$vl = $$rp - $$ip + 1;
          21       *       6              DO VECTOR ($$I2 = $$ip; $$I2 != $$rp; $$I2++) {
          23      25      86                 $$LV_1 = $$LV_1 + DOT_PRODUCT(*(ac[j] + (1 +
($$I2 - 1))*8)
                                    $                 , v[*(ak[j] + $$I2*4)]);
                                      }
                                   }
           *       *      20           result[j] = $$LV_1;
          19      42       6           j = j + 1;
                                   }

                                Vector Parallel Statements

        Stmt Parallel Choices  Chosen      Reason  Vector Choices  Chosen      Reason

          25                2       2                              2       2    (15, 11)

      (15)  Loop was split into two loops to get vector and parallel execution.
      (11)  Dependences prevent vectorization and parallelization of some loops.


                            Non-vector Non-parallel Statements

      Following statements have been neither vectorized nor parallelized:

       Stmt        Reason     Stmt        Reason     Stmt        Reason
```

```
   42          (11)

(11)  Dependences prevent vectorization and parallelization of some loops.


Following scalar variables create problem dependences:
     j

Following vector ops in line 23 run at scalar speed
                    MUL (integer), CONVERT (integer), DIV (integer, integer)


                Vectorized Results From File vecxvec.c
                      Origin -- Line 13

 Line   Stmt   Time        Program

    *      *      3        $$ds = 32;
    *      *      6        if ($$B1 < 96 & $$B1 >= 4) $$ds = $$B1>>2;
    *      *      8        $$LV_1 = result;
   13      *      9        DO PARALLEL ($$ip = 0; $$ip != (int) (dim) - 1; $$ip +=
$$ds) {
   13      *     15           $$rp = MIN((int) (dim) - 1, $$ip - 1 + $$ds);
    *      *      9           $$vl = $$rp - $$ip + 1;
   13      *      6           DO VECTOR ($$I1 = $$ip; $$I1 != $$rp; $$I1++) {
   15      7     40              $$LV_1 = $$LV_1 + DOT_PRODUCT(a[$$I1], b[$$I1]);
                              }
                           }
    *      *      8        result = $$LV_1;


                        Vector Parallel Statements

 Stmt Parallel Choices  Chosen      Reason  Vector Choices  Chosen      Reason

    7                1       1                          1       1        (15)

(15)  Loop was split into two loops to get vector and parallel execution.


                    Vectorized Results From File vxv.c
                          Origin -- Line 13

 Line   Stmt   Time        Program

    *      *      3        $$ds = 32;
    *      *      6        if ($$B1 < 96 & $$B1 >= 4) $$ds = $$B1>>2;
    *      *      8        $$LV_1 = result;
   13      *      9        DO PARALLEL ($$ip = 0; $$ip != (int) (dim) - 1; $$ip +=
$$ds) {
   13      *     15           $$rp = MIN((int) (dim) - 1, $$ip - 1 + $$ds);
    *      *      9           $$vl = $$rp - $$ip + 1;
   13      *      6           DO VECTOR ($$I1 = $$ip; $$I1 != $$rp; $$I1++) {
   15      7     40              $$LV_1 = $$LV_1 + DOT_PRODUCT(a[$$I1], a[$$I1]);
                              }
                           }
    *      *      8        result = $$LV_1;


                        Vector Parallel Statements

 Stmt Parallel Choices  Chosen      Reason  Vector Choices  Chosen      Reason

    7                1       1                          1       1        (15)

(15)  Loop was split into two loops to get vector and parallel execution.
```

```
                    Vectorized Results From File landav.c
                          Origin -- Line 12

       Line    Stmt    Time         Program

          *       *       3         $$ds = 32;
          *       *       6         if ($$B1 < 96 & $$B1 >= 4) $$ds = $$B1>>2;
         12       *       9         DO PARALLEL ($$ip = 0; $$ip != (int) (dim) - 1; $$ip +=
$$ds) {
         12       *      15            $$rp = MIN((int) (dim) - 1, $$ip - 1 + $$ds);
          *       *       9            $$vl = $$rp - $$ip + 1;
         12       *       6            DO VECTOR ($$I1 = $$ip; $$I1 != $$rp; $$I1++) {
         14       7      34               c[$$I1] = landa * v[$$I1];
                                        }
                                     }


                              Vector Parallel Statements

       Stmt Parallel Choices  Chosen      Reason  Vector Choices  Chosen      Reason

          7                1       1                              1       1        (15)

      (15)  Loop was split into two loops to get vector and parallel execution.


                    Vectorized Results From File vecxmat.c
                          Origin -- Line 14

       Line    Stmt    Time         Program

          *       *       3         $$ds = 32;
          *       *       6         if ($$B1 < 96 & $$B1 >= 4) $$ds = $$B1>>2;
         14       *       9         DO PARALLEL ($$ip = 0; $$ip != (int) (columnas) - 1; $$ip
+= $$ds) {
         14       *      15            $$rp = MIN((int) (columnas) - 1, $$ip - 1 + $$ds);
          *       *       9            $$vl = $$rp - $$ip + 1;
         14       *       6            DO VECTOR ($$I1 = $$ip; $$I1 != $$rp; $$I1++) {
         16       8      55               sol[$$I1] = a[0] * *(b[0] + $$I1*8);
                                        }
                                     }


                              Vector Parallel Statements

       Stmt Parallel Choices  Chosen      Reason  Vector Choices  Chosen      Reason

          8                1       1                              1       1    (10, 15)

      (10)  VBEST directive mandated vector loop selection.
      (15)  Loop was split into two loops to get vector and parallel execution.

      Following vector ops in line 16 run at scalar speed
                        MUL (integer), DIV (integer)
      ------------------------------------
                    Vectorized Results From File vecxmat.c
                          Origin -- Line 18

       Line    Stmt    Time         Program

          *      22       7         $$B2 = MAX((int) (columnas), 0);
         20       *       8         DO PARALLEL ($$I2 = 0; $$I2 != (int) (columnas) - 1;
$$I2++) {
         18       *       8            for ($$iv = 1; $$iv != (int) (filas) - 1; $$iv += 32)
{
         18       *      12               $$rv = MIN((int) (filas) - 1, 31 + $$iv);
          *       *       9               $$vl = $$rv - $$iv + 1;
          *       *       9               $$vl = $$rv - $$iv + 1;
         18       *       6               DO VECTOR ($$I1 = $$iv; $$I1 != $$rv; $$I1++) {
         23      26      73                  sol[$$I2] = sol[$$I2] + DOT_PRODUCT(a[$$I1],
*(b[$$I1]
                                     $                + $$I2*8));
                                        }
                                     }
                                  }


                              Vector Parallel Statements
```

```
 Stmt Parallel Choices  Chosen      Reason  Vector Choices  Chosen      Reason

   26                2       1                       1, 2       2


Following vector ops in line 23 run at scalar speed
                    DIV (integer)

                Vectorized Results From File solvey.c
                       Origin -- Line 17

 Line   Stmt   Time        Program

   17      *      8        for ($$I1 = 0; $$I1 != (int) (filas) - 1; $$I1++) {
   19      8     38           sol[1 + j] = *a[j];
    *     11      7           $$B2 = MAX((int) (j), 0);
    *      *      3           $$ds = 32;
    *      *      6           if ($$B2 < 96 & $$B2 >= 4) $$ds = $$B2>>2;
    *      *     22           $$LV_1 = sol[1 + j];
   21      *      9           DO PARALLEL ($$ip = 0; $$ip != (int) (j) - 1; $$ip +=
$$ds) {
   21      *     15              $$rp = MIN((int) (j) - 1, $$ip - 1 + $$ds);
    *      *      9              $$vl = $$rp - $$ip + 1;
   21      *      6              DO VECTOR ($$I2 = $$ip; $$I2 != $$rp; $$I2++) {
   23     15     59                 $$LV_1 = $$LV_1 - DOT_PRODUCT(*(a[j] + ($$I2 +
1)*8), sol[1 + $$I2                     ]);
                                  }
                               }
    *      *     22           sol[1 + j] = $$LV_1;
   25     24     92           sol[1 + j] = sol[1 + j] / *(a[j] + (j + 1)*8);
   17     36      6           j = j + 1;
                           }

                        Vector Parallel Statements

 Stmt Parallel Choices  Chosen      Reason  Vector Choices  Chosen      Reason

   15                2       2                       2       2    (15, 11)

(15)  Loop was split into two loops to get vector and parallel execution.
(11)  Dependences prevent vectorization and parallelization of some loops.

IVDEP influenced construction of array graph

                    Non-vector Non-parallel Statements

Following statements have been neither vectorized nor parallelized:

 Stmt         Reason     Stmt         Reason     Stmt         Reason

    8           (11)       24           (11)       36           (11)

(11)  Dependences prevent vectorization and parallelization of some loops.

IVDEP influenced construction of array graph.
Following scalar variables create problem dependences:
     j

The compiler assumes that the following dependences exist
because it cannot prove they do not.  If you know they do
not exist, IVDEP or IPDEP will improve the resulting code.

 From      To  From Expr                To Expr
   24      24  sol[1 + j]               sol[1 + j]
   24      24  sol[1 + j]               sol[1 + j]
   24       8  sol[1 + j]               sol[1 + j]
   24      24  sol[1 + j]               sol[1 + j]
   24       8  sol[1 + j]               sol[1 + j]
    8      24  sol[1 + j]               sol[1 + j]
    8      24  sol[1 + j]               sol[1 + j]
    8       8  sol[1 + j]               sol[1 + j]
Following vector ops in line 23 run at scalar speed
                    MUL (integer), DIV (integer)
----------------------------------------
```

```
                    Vectorized Results From File solvey.c
                          Origin -- Line 28

 Line   Stmt   Time        Program

   *      *      3        $$ds = 32;
   *      *      6        if ($$B1 < 96 & $$B1 >= 4) $$ds = $$B1>>2;
  28      *      9        DO PARALLEL ($$ip = 0; $$ip != (int) (filas) - 1; $$ip +=
$$ds) {
  28      *     15           $$rp = MIN((int) (filas) - 1, $$ip - 1 + $$ds);
   *      *      9           $$vl = $$rp - $$ip + 1;
  28      *      6           DO VECTOR ($$I1 = $$ip; $$I1 != $$rp; $$I1++) {
  30     44     31              preres[1 + $$I1] = sol[$$I1];
                             }
                          }

                          Vector Parallel Statements

 Stmt Parallel Choices  Chosen      Reason  Vector Choices  Chosen      Reason

   44                1       1                          1       1        (15)

(15)  Loop was split into two loops to get vector and parallel execution.

IVDEP influenced construction of array graph
-------------------------------------------
                    Vectorized Results From File solvey.c
                          Origin -- Line 34

 Line   Stmt   Time        Program

   34     55      4        do
                           {
   34     56      6           $$B1 = $$B1 + 1;
   36     58      6           j = j - 1;
    *     61     13           $$B2 = MAX((int) (filas) - (int) (1 + j), 0);
   38     62      4           $$R2 = $$B2;
    *      *      3           $$ds = 32;
    *      *      6           if ($$B2 < 96 & $$B2 >= 4) $$ds = $$B2>>2;
    *      *     20           $$LV_2 = sol[j];
   38      *      6           DO PARALLEL ($$ip = 1; $$ip != $$R2; $$ip += $$ds) {
   38      *     12              $$rp = MIN($$R2, $$ip - 1 + $$ds);
    *      *      9              $$vl = $$rp - $$ip + 1;
   38      *      6              DO VECTOR ($$I2 = $$ip; $$I2 != $$rp; $$I2++) {
   40     65     61                 $$LV_2 = $$LV_2 - DOT_PRODUCT(*(a[j + $$I2] + (j
+ 1)*8)
                                $                 , sol[j + $$I2]);
                                   }
                               }
    *      *     20              sol[j] = $$LV_2;
   43     74     88              sol[j] = sol[j] / *(a[j] + (j + 1)*8);
                              } while (j > 0);

   34      *      4        $$R1 = $$B1;

                          Vector Parallel Statements

 Stmt Parallel Choices  Chosen      Reason  Vector Choices  Chosen      Reason

   65                2       2                          2       2    (15, 11)

(15)  Loop was split into two loops to get vector and parallel execution.
(11)  Dependences prevent vectorization and parallelization of some loops.

IVDEP influenced construction of array graph

                          Non-vector Non-parallel Statements

Following statements have been neither vectorized nor parallelized:

 Stmt        Reason     Stmt        Reason     Stmt        Reason

   58          (11)       74          (11)

(11)  Dependences prevent vectorization and parallelization of some loops.
```

```
IVDEP influenced construction of array graph.
Following scalar variables create problem dependences:
     j

The compiler assumes that the following dependences exist
because it cannot prove they do not.  If you know they do
not exist, IVDEP or IPDEP will improve the resulting code.


 From     To  From Expr                To Expr
   74     74  sol[j]                   sol[j]
   74     74  sol[j]                   sol[j]
   74     74  sol[j]                   sol[j]
Following vector ops in line 40 run at scalar speed
                    DIV (integer)
-------------------------------------
             Vectorized Results From File solvey.c
                    Origin -- Line 49

 Line   Stmt   Time        Program

    *      *      3        $$ds = 32;
    *      *      6        if ($$B1 < 96 & $$B1 >= 4) $$ds = $$B1>>2;
    *      *      8        $$LV_3 = tmp;
   49      *      9        DO PARALLEL ($$ip = 0; $$ip != (int) (filas) - 1; $$ip +=
$$ds) {
   49      *     15           $$rp = MIN((int) (filas) - 1, $$ip - 1 + $$ds);
    *      *      9           $$vl = $$rp - $$ip + 1;
   49      *      6           DO VECTOR ($$I1 = $$ip; $$I1 != $$rp; $$I1++) {
   51     89     44              $$LV_3 = $$LV_3 + DOT_PRODUCT(*a[$$I1], sol[$$I1]);
                               }
                            }
    *      *      8        tmp = $$LV_3;

                        Vector Parallel Statements

 Stmt Parallel Choices  Chosen      Reason  Vector Choices  Chosen      Reason

   89               1       1                           1       1        (15)

(15)  Loop was split into two loops to get vector and parallel execution.

Following vector ops in line 51 run at scalar speed
                    DIV (integer)
-------------------------------------
             Vectorized Results From File solvey.c
                    Origin -- Line 55

 Line   Stmt   Time        Program

    *      *      3        $$ds = 32;
    *      *      6        if ($$B1 < 96 & $$B1 >= 4) $$ds = $$B1>>2;
   55      *      9        DO PARALLEL ($$ip = 0; $$ip != (int) (filas) - 1; $$ip +=
$$ds) {
   55      *     15           $$rp = MIN((int) (filas) - 1, $$ip - 1 + $$ds);
    *      *      9           $$vl = $$rp - $$ip + 1;
   55      *      6           DO VECTOR ($$I1 = $$ip; $$I1 != $$rp; $$I1++) {
   57    103     71              sol[$$I1] = sol[$$I1] * (beta/(1.0D0 + tmp));
                               }
                            }

                        Vector Parallel Statements

 Stmt Parallel Choices  Chosen      Reason  Vector Choices  Chosen      Reason

  103               1       1                           1       1        (15)

(15)  Loop was split into two loops to get vector and parallel execution.


-------------------------------------
             Vectorized Results From File solvey.c
                    Origin -- Line 64

 Line   Stmt   Time        Program
```

```
          *       *       3        $$ds = 32;
          *       *       6        if ($$B1 < 96 & $$B1 >= 4) $$ds = $$B1>>2;
         64       *       7        DO PARALLEL ($$ip = 1; $$ip != (int) (filas); $$ip +=
$$ds) {
         64       *      13           $$rp = MIN((int) (filas), $$ip - 1 + $$ds);
          *       *       9           $$vl = $$rp - $$ip + 1;
         64       *       6           DO VECTOR ($$I1 = $$ip; $$I1 != $$rp; $$I1++) {
         66     119      83              preres[$$I1] = preres[$$I1] * (-beta/(1.0D0 +
tmp));
                                      }
                                   }
```

Vector Parallel Statements

Stmt	Parallel	Choices	Chosen	Reason	Vector	Choices	Chosen	Reason
119		1	1			1	1	(15)

(15) Loop was split into two loops to get vector and parallel execution.

Vectorized Results From File upluseqv.c
Origin -- Line 11

```
Line    Stmt    Time        Program

   •       •       3        $$ds = 32;
   •       •       6        if ($$B1 < 96 & $$B1 >= 4) $$ds = $$B1>>2;
  11       •       9        DO PARALLEL ($$ip = 0; $$ip != (int) (dim) - 1; $$ip +=
$$ds) {
  11       *      15           $$rp = MIN((int) (dim) - 1, $$ip - 1 + $$ds);
   *       *       9           $$vl = $$rp - $$ip + 1;
  11       *       6           DO VECTOR ($$I1 = $$ip; $$I1 != $$rp; $$I1++) {
  13       7      44              u[$$I1] = u[$$I1] + v[$$I1];
                                  }
                               }
```

Vector Parallel Statements

Stmt	Parallel	Choices	Chosen	Reason	Vector	Choices	Chosen	Reason
7		1	1			1	1	(15)

(15) Loop was split into two loops to get vector and parallel execution.

Vectorized Results From File umenosv.c
Origin -- Line 12

```
Line    Stmt    Time        Program

   *       *       3        $$ds = 32;
   *       *       6        if ($$B1 < 96 & $$B1 >= 4) $$ds = $$B1>>2;
  12       *       9        DO PARALLEL ($$ip = 0; $$ip != (int) (dim) - 1; $$ip +=
$$ds) {
  12       *      15           $$rp = MIN((int) (dim) - 1, $$ip - 1 + $$ds);
   *       *       9           $$vl = $$rp - $$ip + 1;
  12       *       6           DO VECTOR ($$I1 = $$ip; $$I1 != $$rp; $$I1++) {
  14       7      44              c[$$I1] = u[$$I1] - v[$$I1];
                                  }
                               }
```

Vector Parallel Statements

Stmt	Parallel	Choices	Chosen	Reason	Vector	Choices	Chosen	Reason
7		1	1			1	1	(15)

(15) Loop was split into two loops to get vector and parallel execution.

==========

Como podemos observar claramente, el compilador es capaz de emitir un código de tipo vectorial/paralelo ("DO VECTOR", "DO PARALLEL") en las subrutinas esenciales, siendo necesario instruirle mediante directivas ("IVDEP", "IPDEP") de la posibilidad de llevarlo a cabo ignorando dependencias aparentes. En otros casos ha sido necesario instruir al compilador sobre la posible elección de ejecución vectorial y/o paralela mediante las directivas "PBEST" y "VBEST"; estas últimas directivas, así como las anteriormente citadas, son compatibles con el conjunto de directivas presentes en el compilador del supercomputador CRAY YMP, lo que hace posible el transporte del código informático a grandes superordenadores de forma inmediata y directa.

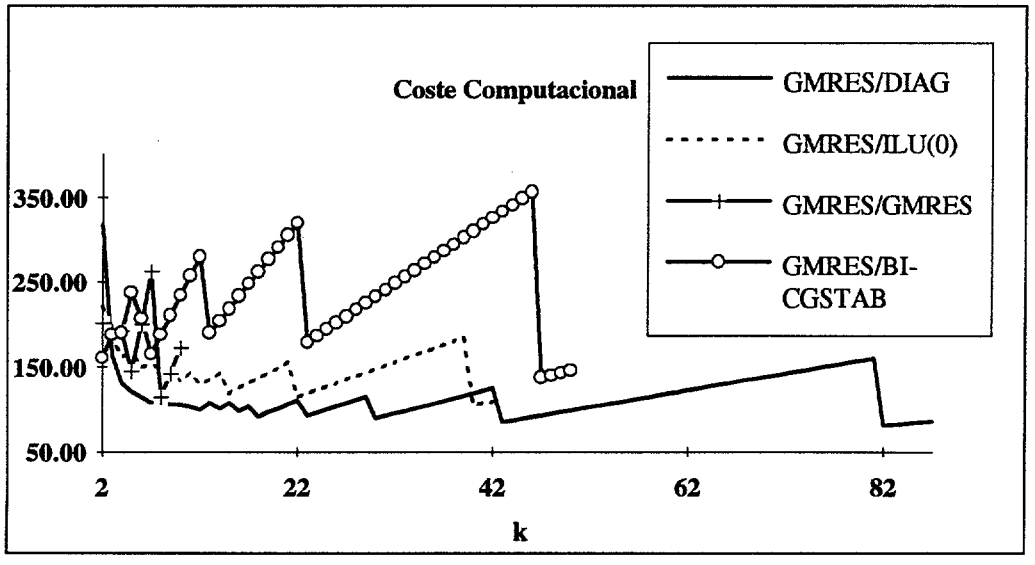
APLICACIONES

APLICACIONES NUMÉRICAS

EVALUACIÓN INICIAL

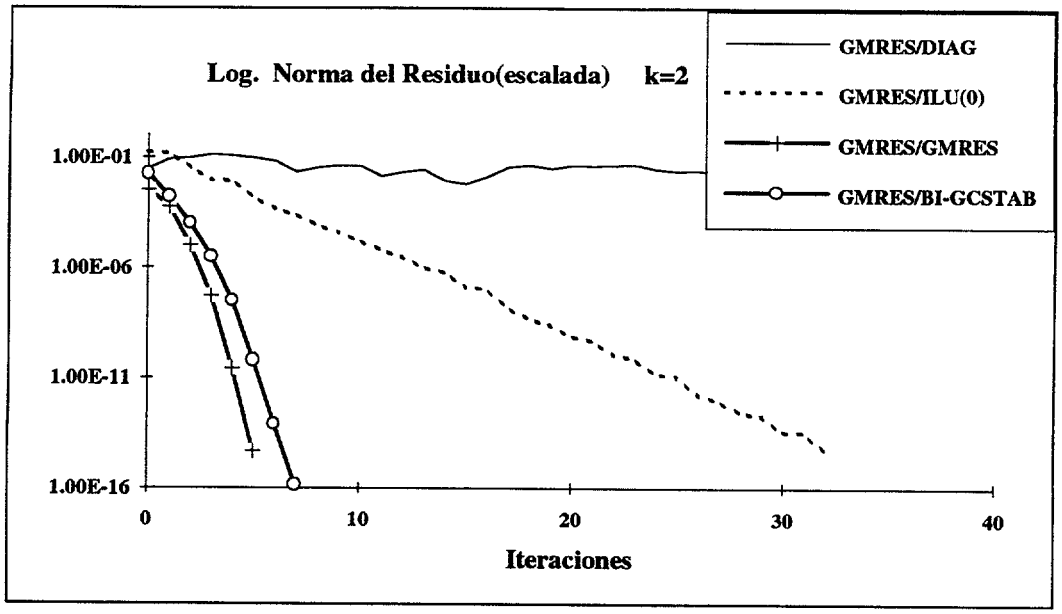
El algoritmo propuesto ha sido aplicado, en primer lugar a matrices generadas de forma aleatoria (ver Marsaglia[28],[29]), con órdenes 1.000 y 100.000, para el estudio del comportamiento de los precondicionadores.

Observamos el comportamiento del GMRES(k) para distintos valores de la dimensión del espacio de Krylov para un valor fijado de la tolerancia (tol=1e-7)

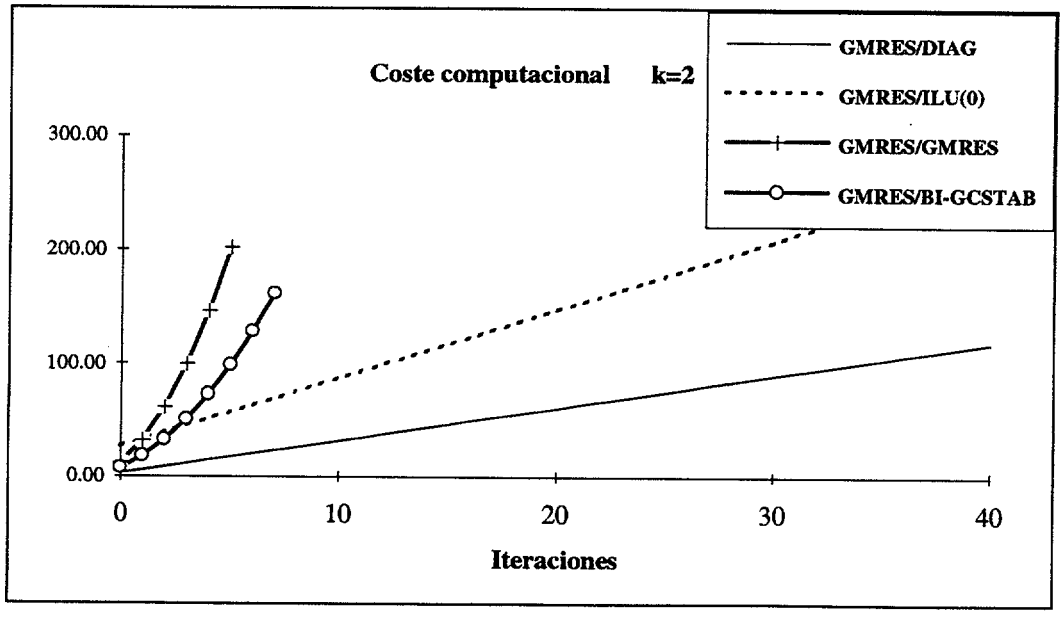


Se observa la gran influencia que tiene el valor de la dimensión del espacio de Krylov en el coste total del algoritmo.

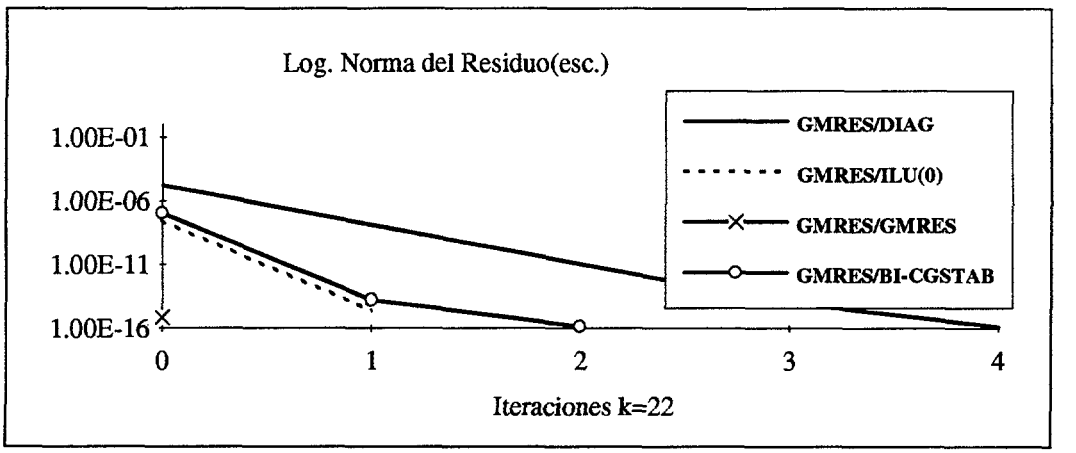
Analizamos con más detalle el efecto de los diversos precondicionadores en el comportamiento del GMRES(k)



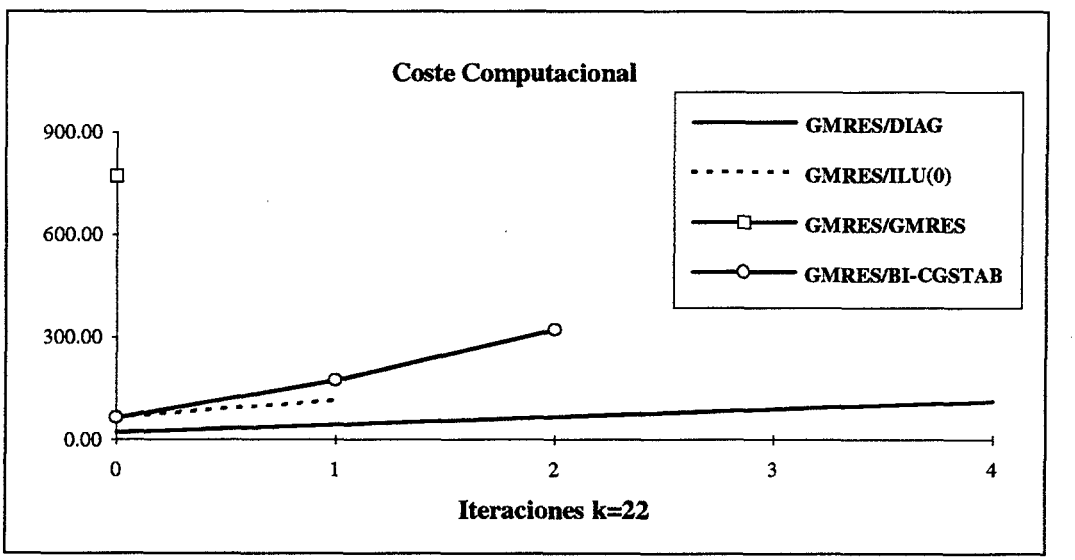
Distribuciones de error con GMRES(k) usando varios precondicionadores (1000 incógnitas)



Tiempos de cálculo del GMRES(k) usando varios precondicionadores (1000 incógnitas).

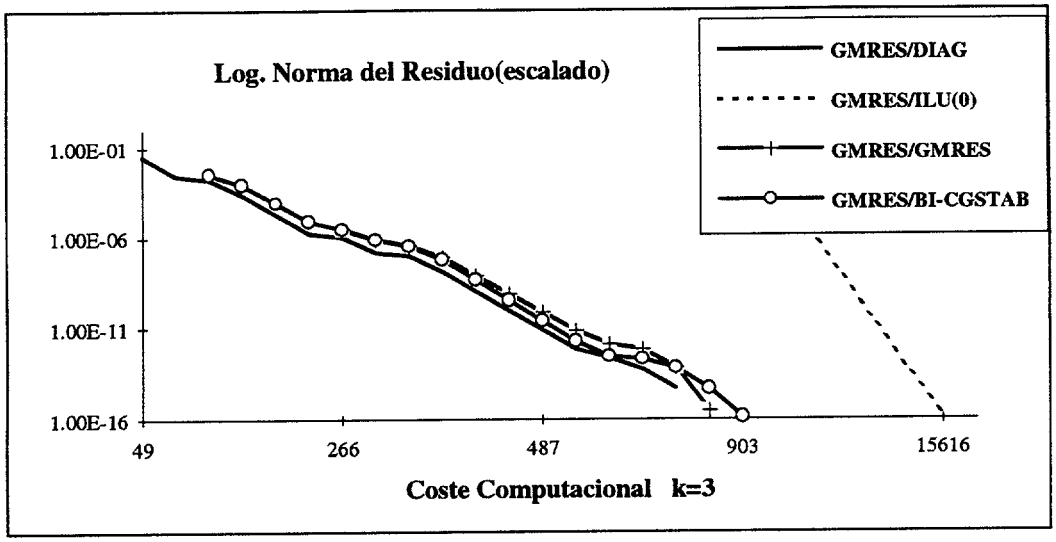


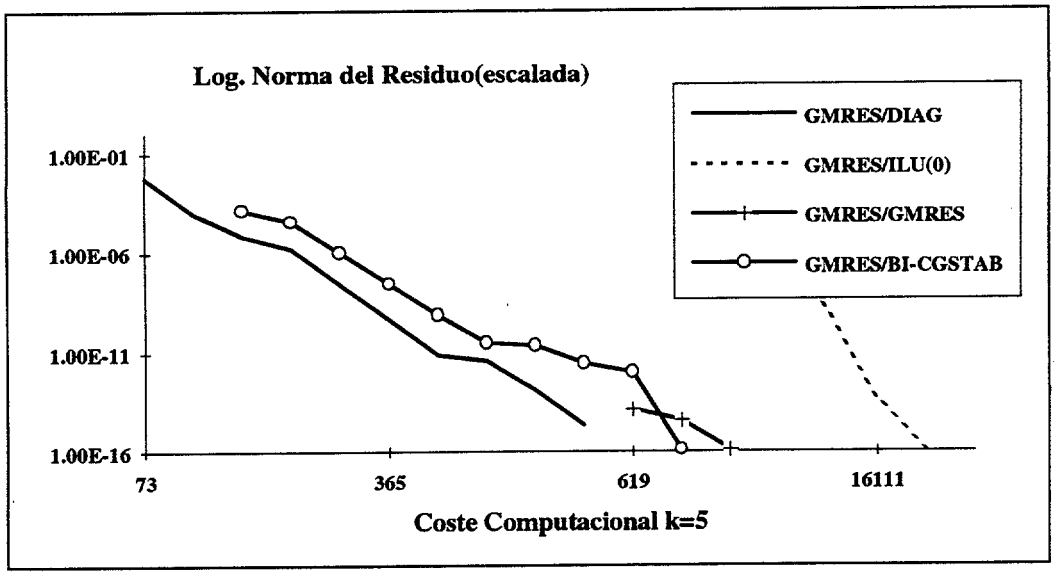
Distribuciones de error con GMRES(k) usando varios precondicionadores (1000 incógnitas)

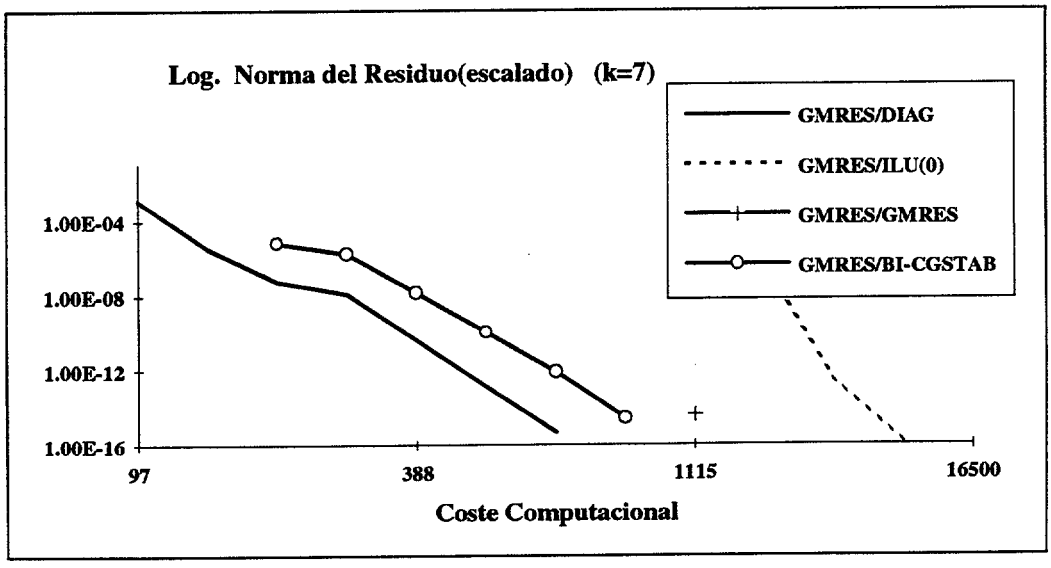


Tiempos de cálculo del GMRES(k) usando varios precondicionadores (1000 incógnitas).

Presentamos varios graficos de coste computacional Usando el GMRES(k), con varios precondicionadores, para un sistema de 100.000 incógnitas y para distintos valores de k





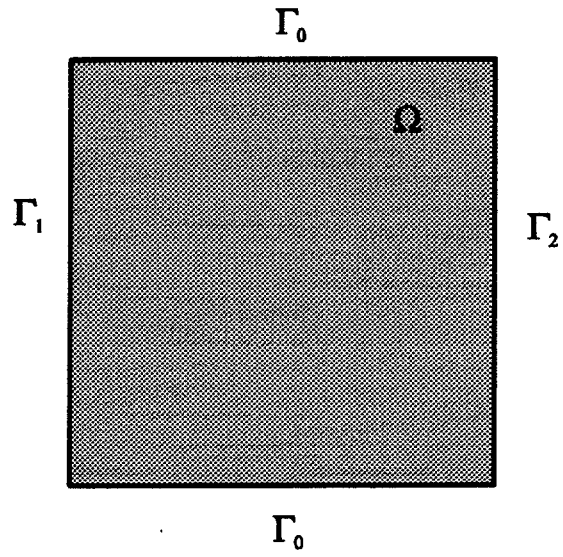


Es notable el alto coste inicial de la factorización ILU(0), sin embargo, cuando el perfil de la mariz es simétrico, se puede reducir este coste inicial de forma substancial.

De estos experimentos numéricos, parece claro el que el precondicionador ILU(0) parece conseguir los mejores resultados de convergencia.

CONVECCIÓN-DIFUSIÓN EN 2D

Se aplica ahora a los sistemas obtenidos de la implementación por el Método de los Elementos Finitos de un problema de Convección-Difusión en un dominio cuadrado (ver Ferragut[16], Montenegro[32]).



Dominio del problema "test"

$$\vec{v}\cdot\vec{\nabla}u-\vec{\nabla}\cdot(k\vec{\nabla}u)=0 \;\; in\;\Omega$$

$$\frac{\delta u}{\delta n} = 0 \ \ en\ \Gamma_0; \quad u = 1 \ \ en\ \Gamma_1; \quad u = 0 \ \ en\ \ \Gamma_2$$

las velocidades vienen dadas por:

$$v_1 = C_v(y - \frac{1}{2})(x - x^2);\ v_2 = C_v(\frac{1}{2} - x)(y - y^2);\ C_v = 10.000$$

Se obtienen sistemas de órdenes 4095(I) y 16333(II), que se resuelven usando el GMRES(variable) cuyos resultados se muestran en las siguientes figuras según los casos:

CASO (a) $Tol' = Tol^{\frac{2}{3}}$ CASO (b) $Tol' = Tol^{\frac{1}{2}}$

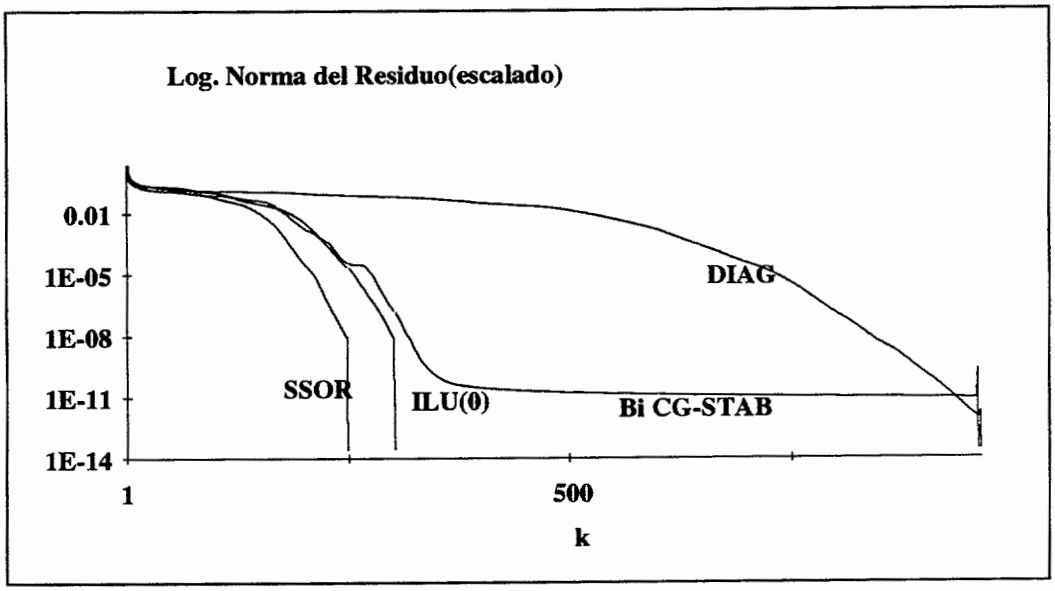
CASO (c) $Tol' = Tol^{\frac{2}{5}}$ CASO (d) $Tol' = Tol$ GMRES ord.

De las gráficas que, a continuación mostraremos, se obtienen algunas indicaciones del comportamiento de los precondicionadores en la resolución por el método GMRES; aunque estas conclusiones, obviamente, no son extrapolables al caso general, pueden servirnos de guía a la hora de abordar problemas de mayor magnitud.

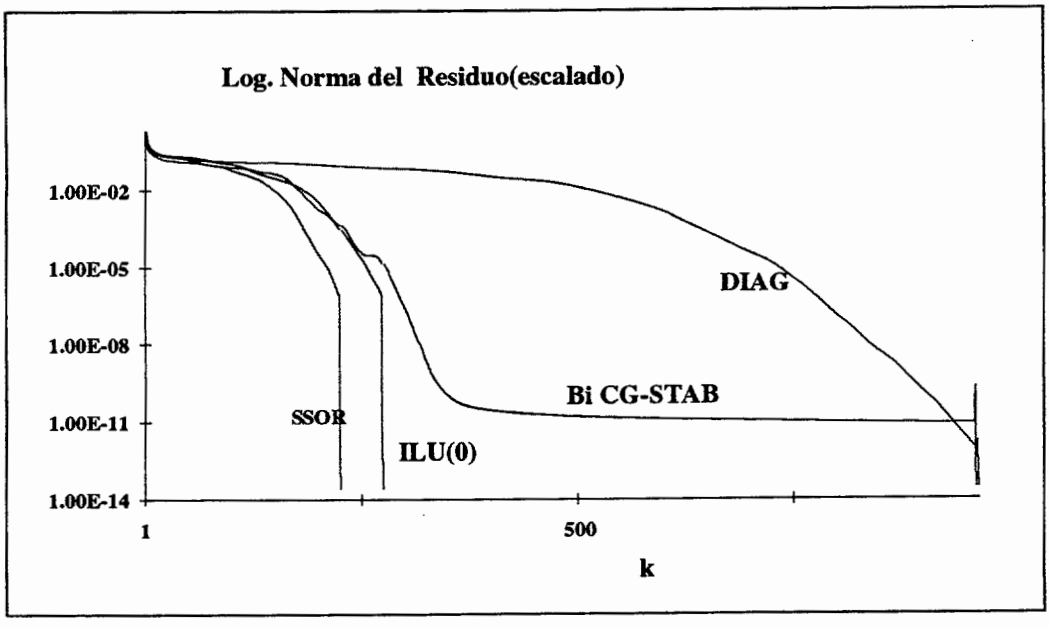
Podremos observar que el comportamiento precondicionador del Bi-CGStab sufre una brusca transformación al exigirse tolerancias más estrictas debido, quizá, a la convergencia de tipo más débil del FGMRES con precondicionador estrictamente variable (ver Saad[41]), también se aprecia el mayor coste, por iteración, de los precondicionadores SSOR e ILU(0) que queda compensado por la gran aceleración de la convergencia que producen. Es de destacar, así mismo, el efecto, confirmado por otros experimentos numéricos, del aumento de la eficiencia del precondicionador ILU(0) al aumentar el orden del sistema. También queda de manifiesto el gran ahorro de memoria que representa el uso del FGMRES(VARIABLE); que se visualiza por el brusco cambio en los perfiles de las curvas.

Estudio del sistema I

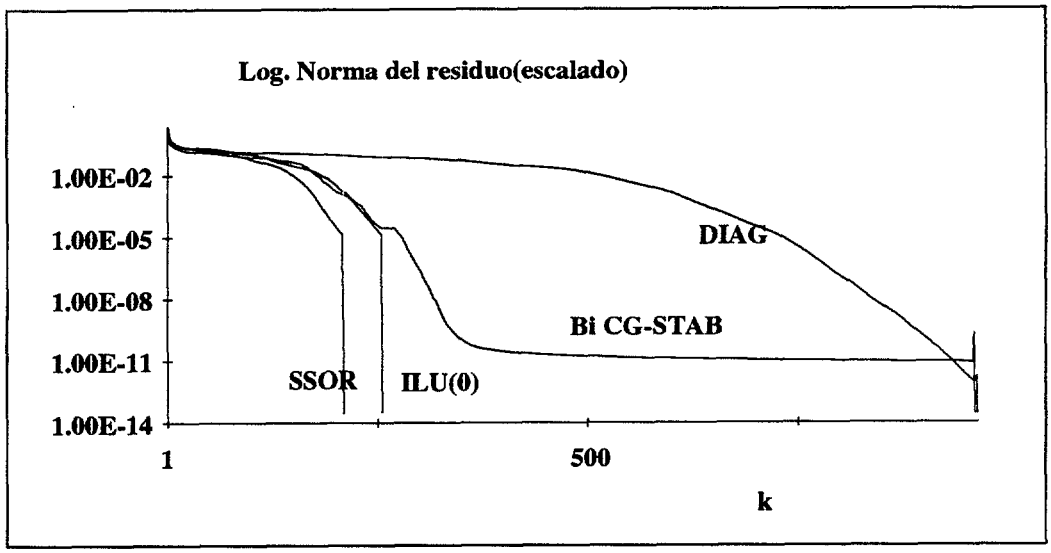
Normas de los residuos en escala logarítmica



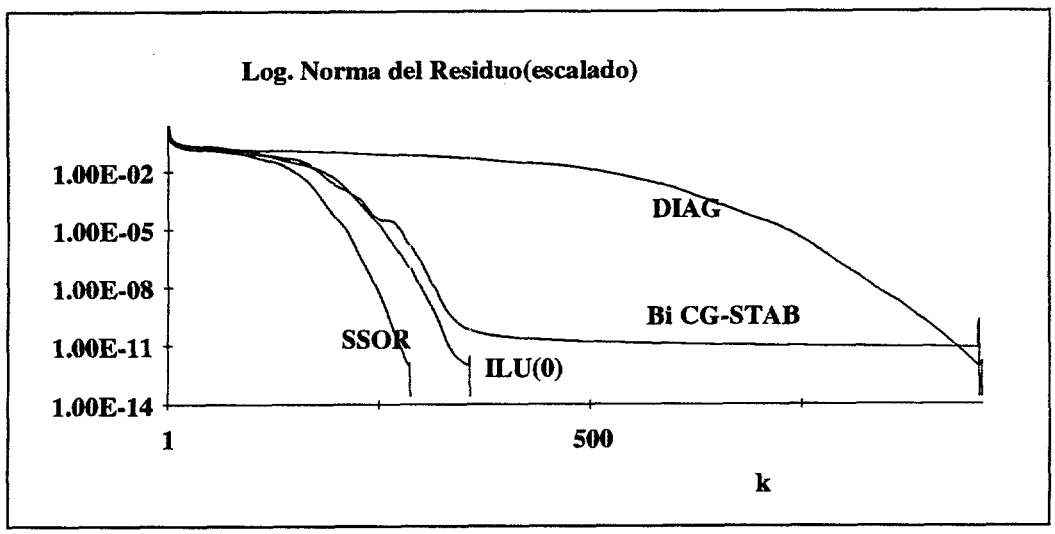
Caso a.



Caso b

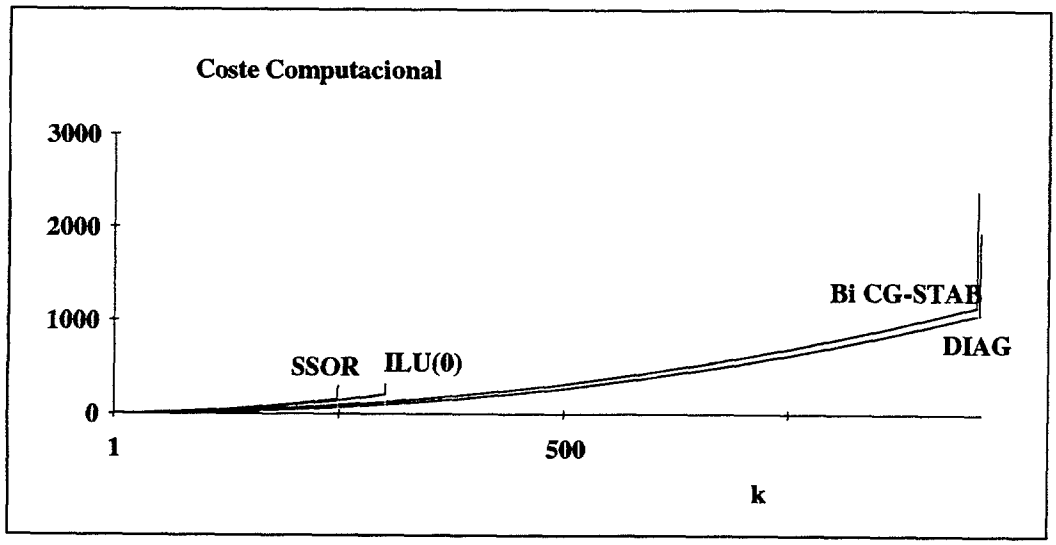


Caso c

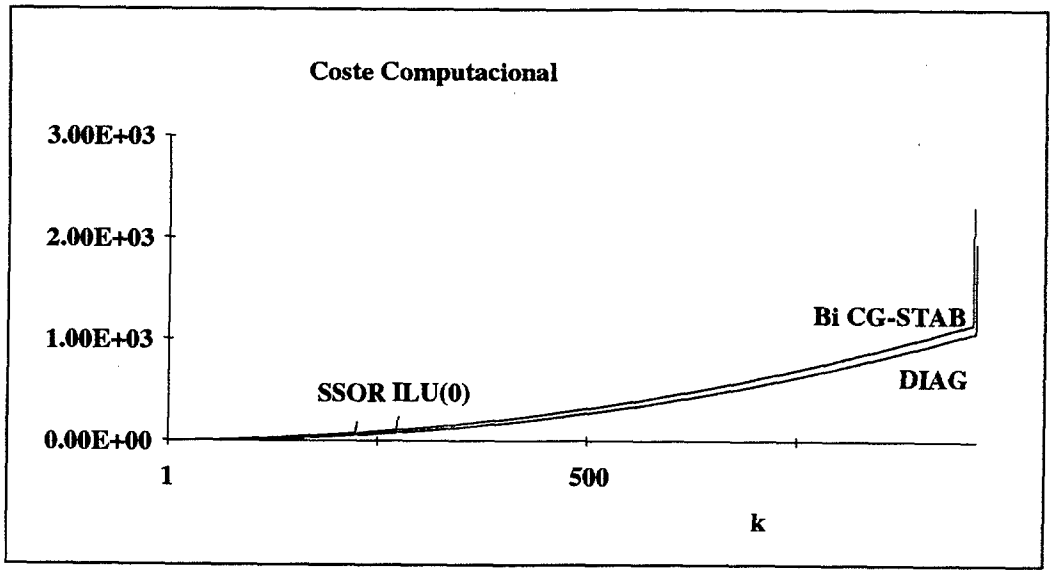


Caso d

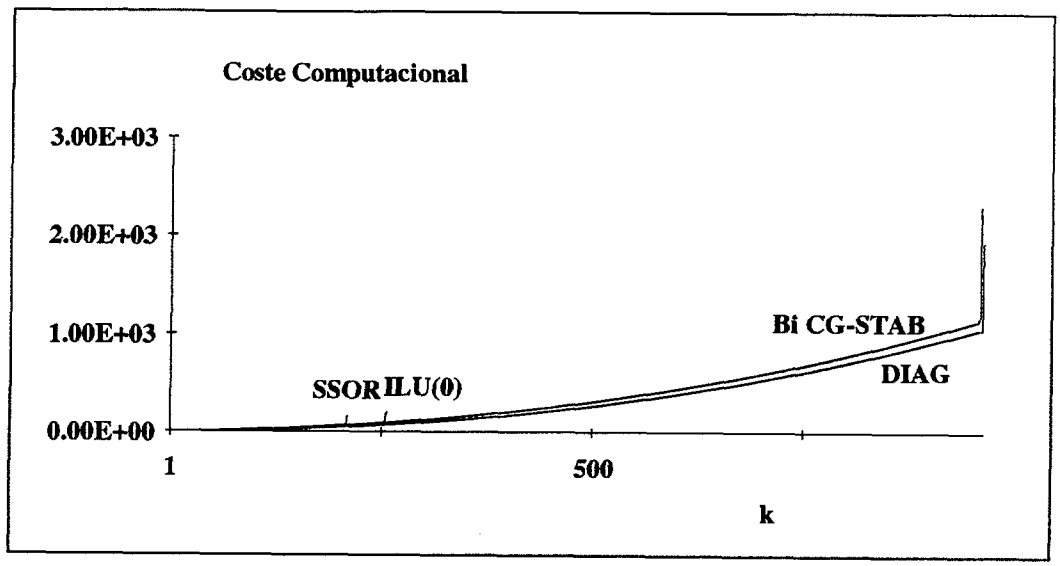
Costes Computacionales



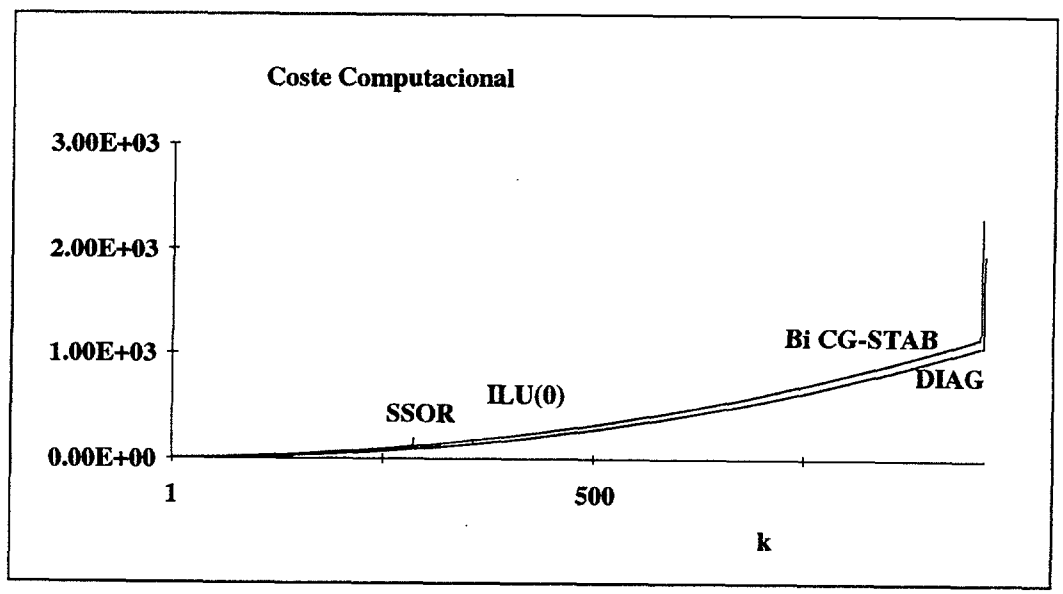
Caso a



Caso b



Caso c

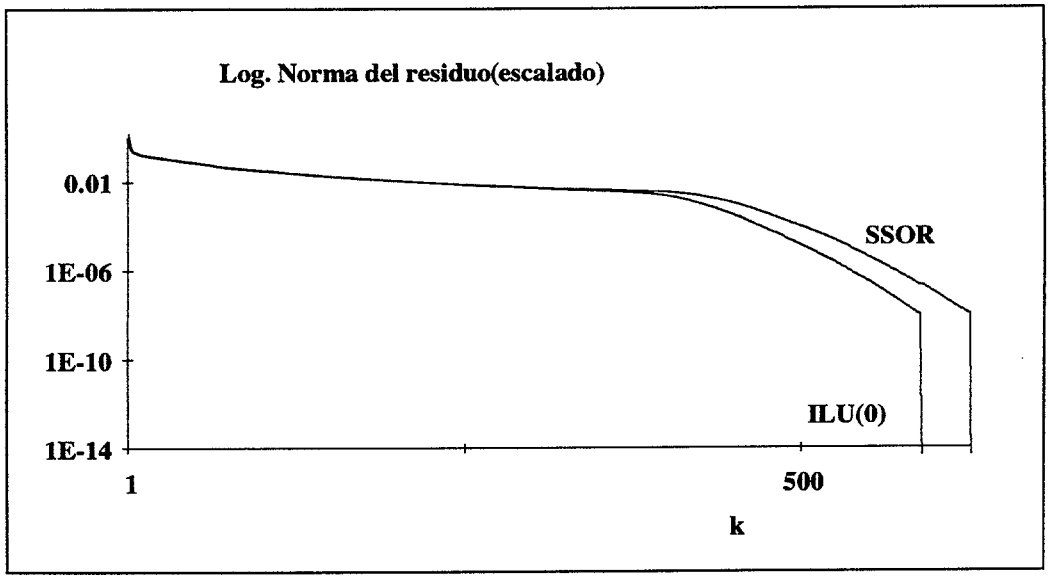


Caso d

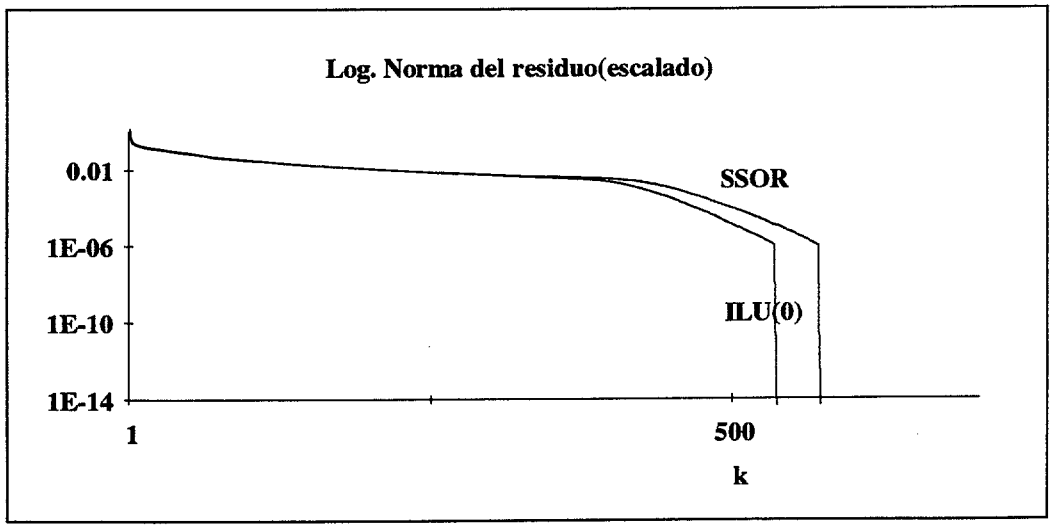
Estudio del sistema II

En este caso., restringimos los precondicionadores unicamente al tipo SSOR e ILU(0).

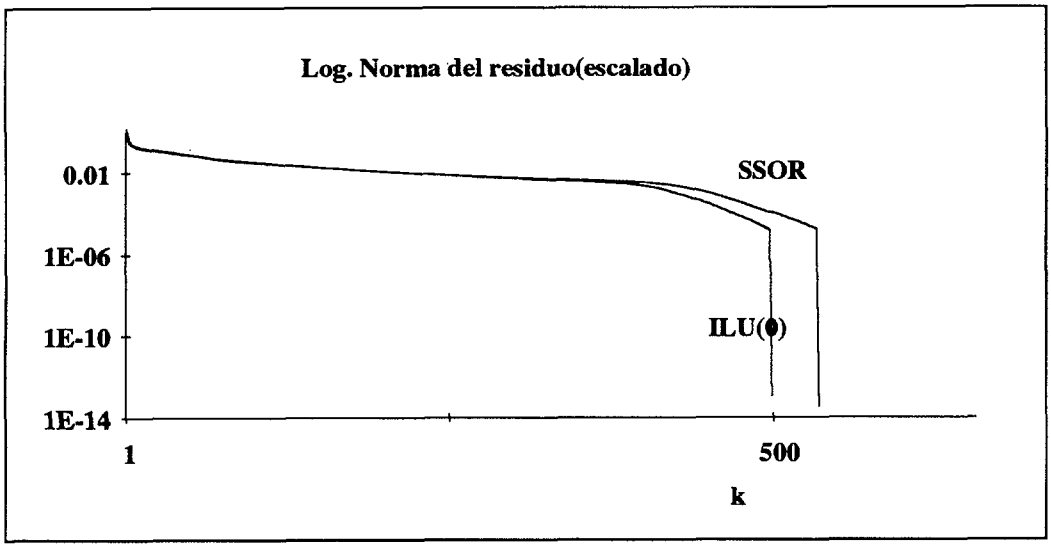
Normas de los residuos en escala logarítmica



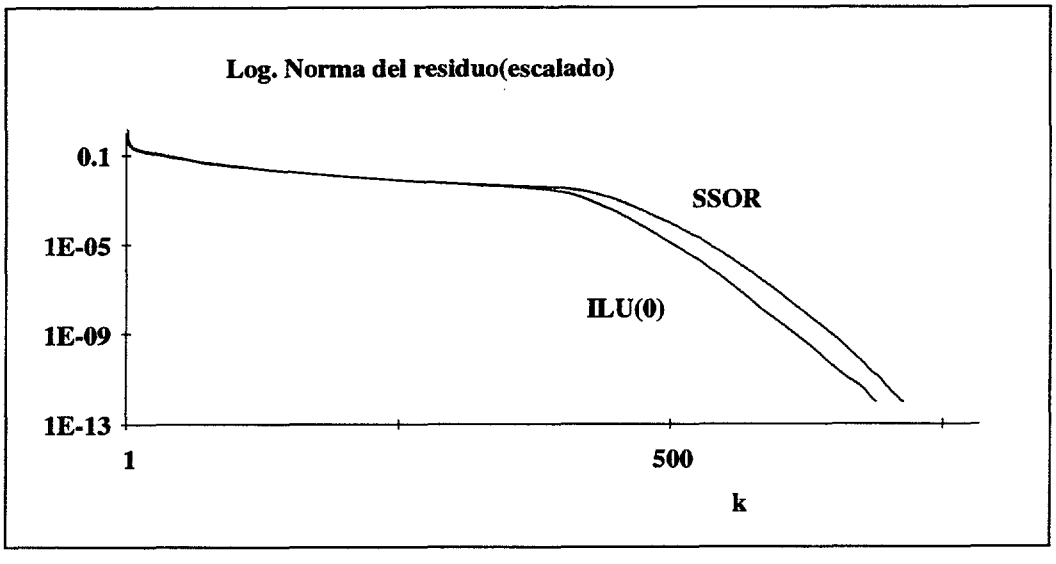
Caso a



Caso b

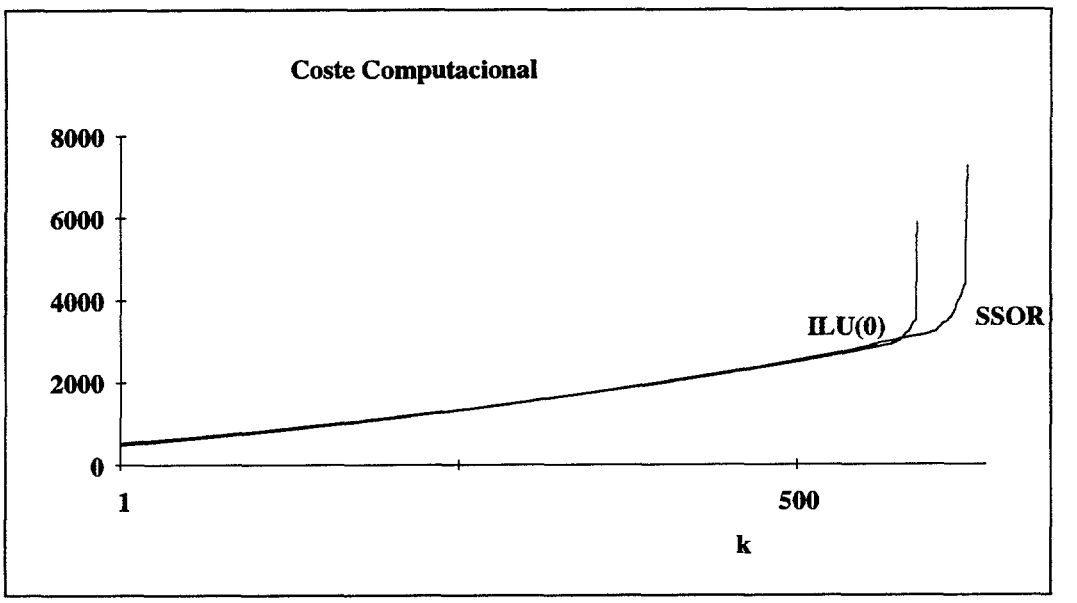


Caso c

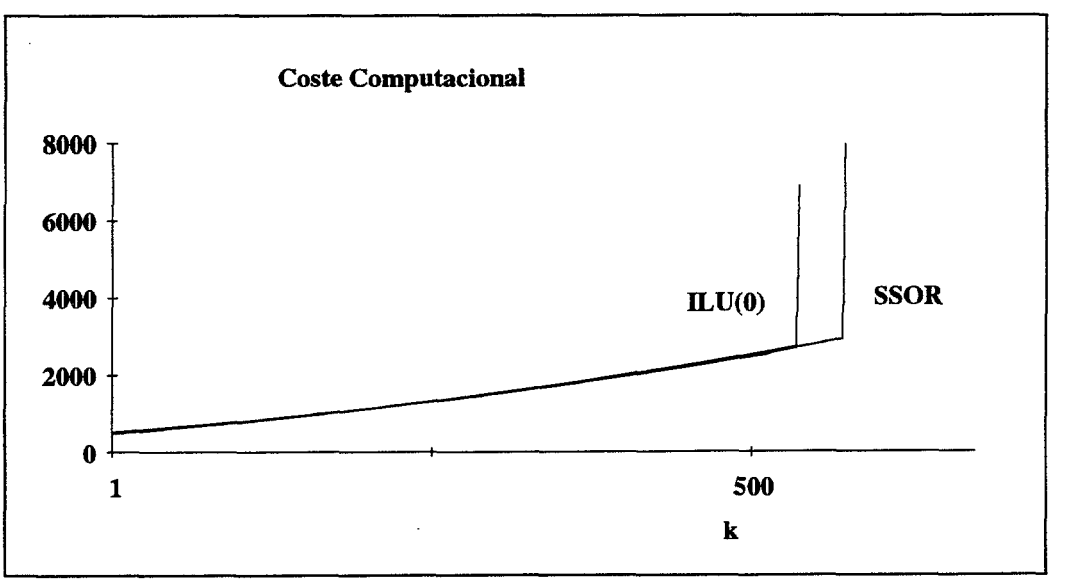


Caso d

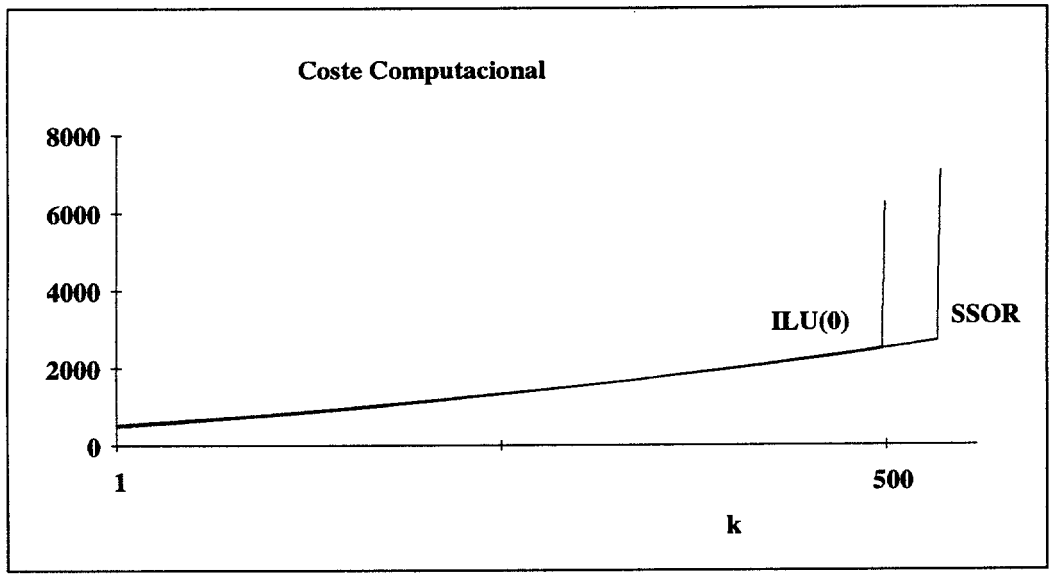
Analizamos ahora los costes computacionales



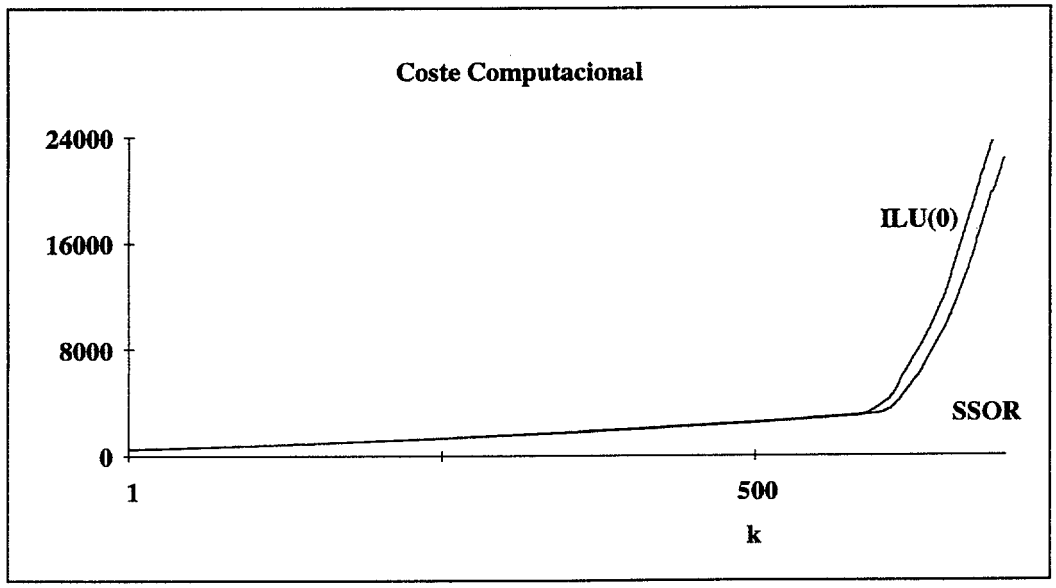
Caso a



Caso b



Caso c



Caso d

La conclusión que podemos obtener de lo anterior es el gran interés que presenta el uso de los precondicionadores SSOR e ILU(0) y el efecto de mejora del ILU(0) con el aumento del orden del sistema.

SIMULACIÓN NUMÉRICA TRIDIMENSIONAL DE CAMPOS DE VIENTO

Para la simulación numérica de campos de vientos en el espacio tridimensional se propone un modelo matemático que se aborda mediante la técnica de elementos finitos.

En este modelo, a partir de un campo inicial de velocidades de viento, se obtiene el campo solución de divergencia nula y verificando la condición de flujo impedido sobre el terreno.

Preproceso de datos in situ e inicialización de velocidades

El movimiento del aire cerca de la superficie del terreno está retrasado por los efectos de fricción proporcionales a la rugosidad de la superficie.

Por tanto, la naturaleza del suelo, la localización y densidad de los árboles, la situación y tamaño de los lagos, ríos, colinas y edificios produce diferentes gradientes de velocidad de viento en la dirección vertical (ver Dickerson[13]).

La capa límite planetaria, es decir, la masa de aire influida por esta fricción se extiende desde algunos cientos de metros hasta varios kilómetros por encima de la superficie del suelo. La altura de esta capa límite es mayor en el caso de condiciones inestables que en caso de condiciones estables.

Debido al cambio apreciable en la velocidad del viento con la altura, dentro de la capa límite planetaria, cualquier valor de la velocidad del viento se deberá citar haciendo referencia a la elevación en que fue medida. La altura internacional convenida para las mediciones del viento superficial es de 10 m.

En nuestro caso, se dispone de diversas medidas de velocidad de viento dadas por estaciones situadas estratégicamente tal que sean representativas de los valores de la velocidad en la región estudiada.

Evidentemente, es necesario que un número mínimo de estaciones estén disponibles. Estas estaciones ubicadas a 10 m. de altura suministran valores de velocidades de viento horizontales, así como su dirección, de forma periódica o bien son resultados promedios.

El primer paso es interpolar dichos valores sobre toda la superficie de la región a estudiar, considerando, también, que los resultados de la interpolación representan las velocidades horizontales a 10 m. de altura.

La técnica seguida para esta interpolación es la clásica ponderación de los valores medidos inversamente con las distancias al cuadrado desde las estaciones al punto considerado:

$$u_{0i} = \frac{\sum_{j=1}^{k} u_{ji} \frac{1}{d_j^2}}{\sum_{j=1}^{k} \frac{1}{d_j^2}} \qquad i = 1,2$$

Siendo k el número de estaciones de medida, u_{ji} las componente i de la velocidad medida en la estación j, y d_j la distancia desde el punto considerado a la estación j.

Una vez se ha calculado el campo interpolado a 10 m. de la superficie para las componentes horizontales de la velocidad, se pasa a aproximar dichos valores en el centro de gravedad de cada elemento del mallado.

Se han realizado numerosos esfuerzos a fin de desarrollar expresiones analíticas adecuadas que relacionen la velocidad con la altura. Aunque no se dispone aún de ninguna expresión totalmente satisfactoria debido a la complejidad de los fenómenos, en usual en meteorología seguir dos caminos, o bien considerar la ley potencial de Deacon (ver Wark[53]) o bien una ley exponencial (ver Christidis[9]).

Se ha observado que la ley de Deacon es de utilidad para capas límites de hasta varios cientos de metros de profundidad. Esta expresión es:

$$\frac{u}{u_1} = \left(\frac{z}{z_1}\right)^p$$

Donde u es la velocidad del viento a la altura z, u_1 es la velocidad del viento a la altura z_1, y p es un exponente positivo con valor entre 0 y 1.

Cuando la tasa de cambio ambiental es aproximadamente igual al valor adiabático y el suelo es generalmente llano con poca cubierta superficial, se debe seleccionar un valor de p aproximadamente igual a 1/7.

El espesor de la capa límite, y por tanto el perfil de la velocidad del viento, es una función de la estabilidad atmosférica. Por ello, el exponente p debe variar en relación con las características de estabilidad de la atmósfera, como una primera aproximación. Sutton[47] ha sugerido que p se puede relacionar con un parámetro n que es una función de la estabilidad de la atmósfera. Esta relación es:

$$p = \frac{n}{2-n}$$

Los valores de n para las varias condiciones de estabilidad están dados en la tabla siguiente:

Condición de estabilidad	n
Tasa de cambio grande	0.20
Tasa de cambio pequeña o cero	0.25
Inversión moderada	0.33
Inversión grande	0.50

Frost relacionó p con la diferencia de temperatura que encontró entre las elevaciones de 5 y 400 pies. Estos datos se presentan en la tabla siguiente:

ΔT(°F)	p	ΔT(°F)	p
-4 a -2	0.145	2 a 4	0.44
-2.5 a -1.5	0.17	4 a 6	0.53
-2 a 0	0.15	6 a 8	0.63
-1 a 1	0.29	8 a 10	0.72
0 a 2	0.32	10 a 12	0.77

Se puede utilizar la ley potencial para aproximar los perfiles de la velocidad del viento a los diversos terrenos que existen en los problemas reales, encontrándose razonablemente adecuado usar un valor de p de 0.4 para áreas urbanizadas, 0.28 para áreas densamente boscosas, ciudades y suburbios, y 0.16 para terrenos llanos y abiertos, lagos y mares.

Algunos investigadores opinan que, para alturas hasta cientos de pies, una relación logarítmica del tipo $u = c \log z$ es más representativa de la capa de aire cerca del terreno que una expresión de la ley potencial, especialmente para terrenos abiertos.

Con esta cualidad, otra forma de aproximar el perfil de vientos según la altura z es seguir una ley exponencial del tipo:

$$u = u_h\left(1 - e^{-\delta z}\right)$$

Donde u_h es la velocidad considerada constante a partir de una altura h (usualmente se toma h=1000 m.), y δ es un parámetro que depende de las propiedades físicas del aire en la atmósfera y de las condiciones de estabilidad atmosférica (ver Hill[27]).

Después de muchos ensayos con una y otra ley, en este trabajo se ha optado por utilizar el segundo camino, dado por la ley exponencial. En realidad, los resultados obtenidos no difieren excesivamente en ambos casos para valores alrededor de 10 m. hacia arriba, siendo las diferencias más acusadas cuando nos acercamos al terreno y más exactas para la segunda ley.

Finalmente, obtenemos la aproximación de la componente vertical de la velocidad del viento en el centro de gravedad de cada elemento del mallado tridimensional. Como la fuerza de retraso por fricción varía con la altura sobre el suelo, la cantidad de desplazamiento angular varía también con la altura.

La fuerza de fricción en la capa límite tiene un máximo cerca de la superficie de la Tierra y cae esencialmente a cero en la parte superior de la capa límite, donde predomina el viento geostrófico o el viento gradiente. Por tanto, el ángulo de desplazamiento de la dirección del viento debido a la fricción varía desde un valor máximo cerca de la superficie terrestre hasta cero en la parte superior de la capa límite.

Según se mira en dirección al suelo, el viento se desplaza en sentido horario con el aumento en altura dentro de la capa límite. En la presencia de fuertes vientos (> 6m/s) el cambio de dirección de la velocidad u con la altura resulta despreciable en los 100 m. inferiores de la capa límite planetaria. No obstante, cuando la velocidad del viento es menor de 6 m/s, el cambio de dirección podrá ser apreciable.

Los valores de dicho ángulo en el rango de 5° a 15° ocurren sobre el océano, mientras que los valores de 25° a 45° son típicos de bajas velocidades sobre masas del suelo.

En esta aplicación se ha considerado una ley exponencial con ciertas características similares a la utilizada anteriormente, función del valor horizontal de la velocidad de viento que permite simular estos cambios de dirección en la componente vertical (ver Caneill[8]).

FORMULACIÓN DEL PROBLEMA

Para un dominio dado $\Omega \subset R^d \, (d = 2,3)$ con frontera $\Gamma = \Gamma_1 \cup \Gamma_2$, buscamos un campo vectorial u que se aproxime a un campo de velocidades u_0 obtenido de la interpolación de medidas experimentales y que verifique el siguiente problema, que recoge la física del problema, por un lado ley de conservación de la masa y por otro desviación del campo de viento originado por la orografía del terreno:

$$div(u) = 0 \quad en \quad \Omega$$

$$u \cdot n = 0 \quad sobre \quad \Gamma_1 \text{ (Topografía)}$$

En resto del contorno del dominio, obviamente la condición debe ser de flujo libre.

El funcional de ajuste considerado es de tipo mínimos cuadrados (ver Winter[54] ,[55], Caneill[8]),

$$J(u) = \frac{1}{2}\int_{\Omega} (u - u_0)^t \, P(u - u_0) d\Omega + \frac{\beta}{2}\int_{\Gamma_2} n.(u - u_0)^2 \, d\Gamma_2$$

Siendo P una matriz diagonal, cuyos términos pueden ponderar, de modo relativo al orden de las unidades entre las medidas horizontales y componente vertical del vector velocidad en el ajuste mínimos cuadrados.

Entonces, el campo vectorial u será la solución del problema de optimización o minimización mínimos cuadrados:

"Encontrar $u \in K$" que verifica, $\mathrm{J}(u) = \min_{v \in K} \mathrm{J}(v)$

Siendo el espacio de funciones admisibles definido al incorporar las restricciones impuestas al campo de velocidades,de acuerdo a la física del problema, $K = \left\{ v; \operatorname{div} v = 0,\ v \cdot n|_{\Gamma_1} = 0 \right\}$

El problema equivale a la determinación del punto silla del Lagrangiano (ver Fortin[18], Ferragut[17], Raviart[40]):

$$\mathrm{L}(v, q) = \mathrm{J}(v) + \int_{\Omega} q \operatorname{div} v \, d\Omega$$

Concretamente, si $L^2(\Omega)$ es el espacio de funciones de cuadrado integrable y $H^1(\Omega)$ el subespacio de $L^2(\Omega)$ de funciones con primeras derivadas de cuadrado integrable, denotamos:

$$H_{0,\Gamma_2}(\Omega) = \left\{ \varphi \in H^1(\Omega);\ \varphi|_{\Gamma} = 0 \right\}$$

$$H(\operatorname{div}, \Omega) = \left\{ v \in \left(L^2(\Omega) \right)^d;\ \operatorname{div} v \in L^2(\Omega) \right\}$$

E introduciendo el espacio de funciones vectoriales tal que $v \cdot n = 0$ *en* Γ_1, es decir:

$$H_{0,\Gamma_1}(\operatorname{div}, \Omega) = \left\{ v \in H(\operatorname{div}, v);\ \int_{\Gamma} v \cdot n \varphi = 0 \quad \forall \varphi \in H_{0,\Gamma_2}(\Omega) \right\}$$

Buscamos la pareja (u,λ) en $H_{0,\Gamma_1}(div,\Omega)\times L^2(\Omega)$ tal que:

$$L(u,q)\le L(u,\lambda)\le L(v,\lambda)$$

$\forall\ q\in L^2(\Omega)$ y para todo $v\in H_{0,\Gamma_1}(\mathrm{div},\Omega)$, lo cual es representado por:

$$\frac{\partial L(u,\lambda)}{\partial v}=0 \quad y \quad \frac{\partial L(u,\lambda)}{\partial q}=0$$

$$\forall v\in H_{0,\Gamma_1}(\mathrm{div},\Omega)$$

$$\int_\Omega q\,\mathrm{div}\,u=0$$

$$\forall q\in L^2(\Omega)$$

Obteniéndose la siguiente formulación:

$$\int_\Omega v^t P u\,d\Omega+\int_\Omega \lambda\,\mathrm{div}\,v\,d\Omega=\int_\Omega v^t P u_0\,d\Omega-\beta\int_{\Gamma_2} v.n(n.(u-u_O))ds$$

Aplicando la formula de Green a la segunda integral del primer miembro:

$$\int_\Omega v^t P u\,d\Omega-\int_\Omega v\,\mathrm{gra}\,d\,\lambda d\Omega=\int_\Omega v^t P u_0\,d\Omega-\beta\int_{\Gamma_2} v.n(n.(u-u_O))ds-\int_{\partial\Omega} v.n\lambda ds$$

Introduciendo la condición de contorno correspondiente a la presencia de la orografía (no penetrabilidad del aire en frontera sólida, desviación del vector velocidad según su incidencia en el terreno),tendremos,

$$\int_\Omega v^t P u\,d\Omega-\int_\Omega v\,\mathrm{gra}\,d\lambda d\Omega=\int_\Omega v^t P u_0\,d\Omega-\beta\int_{\Gamma_2} v.n(n.(u-u_O))ds-\int_{\Gamma_2} v.n\lambda ds$$

Es decir,

$$\int_{\Omega} v' P(u-u_o)\,d\Omega = \int_{\Omega} v\,\mathrm{gra}\,d\lambda\,d\Omega - \beta\int_{\Gamma_2} v.n(n.(u-u_o)+\frac{\lambda}{\beta})ds$$

Imponiendo la regularidad al multiplicador de Lagrange, $\lambda \in H^1(\Omega)$ e introduciendo las condiciones sobre u, se obtiene:

$$u = u_0 + P^{-1}\vec{\nabla}\lambda \quad en \quad \Omega \qquad (*)$$

Recordando la imposición de incomprensibilidad del aire, tendremos, tomando el operador divergencia,

$$-\vec{\nabla}.\left(P^{-1}\vec{\nabla}\lambda\right) = \vec{\nabla}.u_0 \quad en \quad \Omega$$

$$-P^{-1}\frac{\partial\lambda}{\partial n} = n\cdot u_0 \quad sobre \quad \Gamma_1$$

$\lambda = 0$ *sobre* Γ_2 *si* β *es cero, y en el caso general*:

$$-P^{-1}\frac{\partial\lambda}{\partial n} = \frac{\lambda}{\beta} \quad sobre \quad \Gamma_2$$

FORMULACIÓN VARIACIONAL

Establezcamos la formulación variacional en dimensión infinita para el problema de contorno formulado anteriormente, en relación al multiplicador de Lagrange. Multiplicando por la función test, v, del espacio de funciones admisibles e integrando en el dominio, tendremos (ver Winter[54], [55])

$$-\int_{\Omega} Div(\,P^{-1}\,\mathrm{gra}\,d\lambda)\,v\,d\Omega = \int_{\Omega} Div(u_o)\,v\,d\Omega$$

Aplicando fórmula de Green, de integración por partes,

$$\int_{\Omega} P^{-1}\,\mathrm{gra}\,d\lambda\,\mathrm{gra}\,d\,v\,d\Omega = \int_{\Omega} Div(u_o)\,v\,d\Omega + \oint_{\partial\Omega} P^{-1}v\,\mathrm{gra}\,d\lambda\,ds$$

Descomponiendo la integral de contorno,e introduciendo en cada uno de ellos las condiciones que se verifican en los mismos, tendremos,

$$\int_{\Omega} P^{-1} \operatorname{gra} d\lambda \operatorname{gra} d\nu \, d\Omega = \int_{\Omega} Div(u_O)\, \nu \, d\Omega - \int_{\Gamma_1} n.u_O \, \nu ds - \int_{\Gamma_2} \frac{\lambda}{\beta} \nu \, ds$$

Realizando integración por partes en la integral de dominio del segundo miembro,

$$\int_{\Omega} P^{-1} \operatorname{gra} d\lambda \operatorname{gra} d\nu \, d\Omega = -\int_{\Omega} u_O \operatorname{gra} d\nu \, d\Omega + \int_{\Gamma_1 \bigcup \Gamma_2} n.u_O \, \nu \, ds - \int_{\Gamma_1} n.u_O \, \nu ds - \int_{\Gamma_2} \frac{\lambda}{\beta} \nu \, ds$$

teniéndose finalmente,

$$\int_{\Omega} P^{-1} \operatorname{gra} d\lambda \operatorname{gra} d\nu \, d\Omega + \int_{\Gamma_2} \frac{\lambda}{\beta} \nu \, ds = -\int_{\Omega} u_O \operatorname{gra} d\nu \, d\Omega - \int_{\Gamma_2} n.u_O \, \nu ds$$

La discretización de esta formulación por elementos finitos nos genera el sistema lineal de ecuaciones, resolviendo éste obtendremos los valores de λ en los nodos del mallado.

A partir de la ecuación (*), obtenemos los valores de las tres componentes del campo de viento ,vector u en todo el dominio (en el volumen de aire entre el contorno dado por la topografía (ver Apsimon[3]) y zona en altura límite que se considere, la cual debe ser por encima de las montañas más altas, si bien interesará a efectos del potencial eólico las zonas cercanas al suelo, en especial a 10 metros de altura)

Postproceso de Cálculo

Determinando λ_j en cada nodo del mallado de Ω se determina el campo de viento u (u,v,w) siendo cada componente de acuerdo a la formulación del problema resultante de la minimización :

$$u = u_0 + \frac{1}{p_h}\frac{\partial \lambda}{\partial x} \qquad v = v_0 + \frac{1}{p_h}\frac{\partial \lambda}{\partial y} \qquad \omega = \omega_0 + \frac{1}{p_z}\frac{\partial \lambda}{\partial z}$$

Siendo $\vec{u_0} = (u_0, v_0, \omega_0)$ el campo inicial.

El postproceso se ha llevado a cabo, de forma exacta, pues el procedimiento seguido e implementado en el programa informático ha sido:

1.- Cálculo de λ en el centro de gravedad del elemento finito en coordenadas globales.

2.- Por desarrollos de Taylor expresamos para cada elemento finito en coordenadas espaciales , el valor de λ en el c.d.g en relación a los valores de λ en los nodos del elemento finito.

3.- Se construye dos sistemas de ecuaciones (de orden 3) para determinar finalmente los valores del gradiente de λ

Mallado tridimensional en la Simulación Numérica de problemas con orografías variables o complejas.

En problemas numéricos relacionados con el medio ambiente, donde los dominios a discretizar tienen al menos en uno de sus contornos una dependencia total de la orografía del terreno, se debe elegir cuidadosamente el tipo de malla a utilizar.

La complicación de los códigos tridimensionales, así como su implicación en el elevado coste computacional de cualquier cálculo adicional, obligan a estudiar detenidamente la elección del tipo de discretización a seguir.

Discretización de zonas eólicas en 3-D: Preproceso de datos topográficos

En nuestro caso concurren diversas circunstancias que han contribuido a decantar la elección de la malla.

Generalmente, cuando se da el paso de afrontar la generación de mallas tridimensionales es por que se dispone o se ha estudiado previamente el diseño de mallas en 2-D. Este es ciertamente nuestro caso. La generación de mallas bidimensionales se ha desarrollado en trabajos de investigación previos (ver Cuesta[10],[11], Winter[56]).

El dominio tridimensional a discretizar debe generarse a partir de los datos relativos a la topografía del terreno (ver Orlanski[35]). Por tanto, se ha considerado aquí un volumen paralelepipédico en el cual se ha sustituido la cara inferior por la forma del suelo real. En definitiva, el primer paso a seguir es el levantamiento de una malla bidimensional, adecuadamente generada, interpolando los datos que ofrece la digitalización de planos topográficos de la zona de estudio.

La calidad de esta malla va a repercutir no sólo en la calidad de la malla tridimensional, sino que además de ella depende la fiabilidad de la representación del terreno.

Evidentemente, una excesiva discretización en este paso ocasionará una malla aún mas costosa en tres dimensiones.

Existen varios métodos de interpolación de datos topográficos, aunque en este trabajo, a modo de test, se ha utilizado una fórmula interpoladora similar a la utilizada en los campos de velocidades de viento.

Se pueden usar alternativamente otros tipos de interpolación que permitan una mayor representación más fiel de la orografía del terreno, con menor tiempo de cálculo. Esto supone dedicar un esfuerzo considerable a este campo de investigación.

Metodología y diseño gráfico propuesto: Elección del tipo de discretización

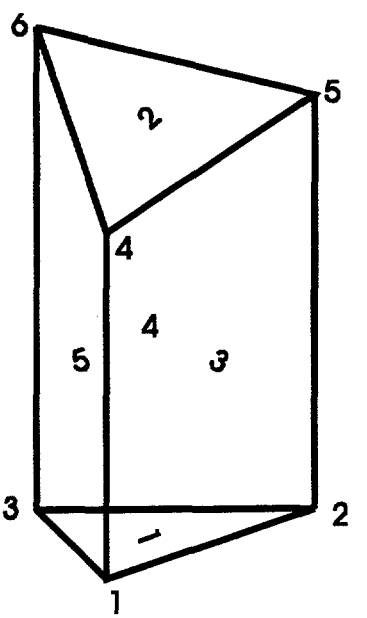
A partir de una malla plana con elementos triangulares y levantada en el espacio, simulando la orografía del terreno (ver Sherman[44]), caben usualmente dos posibilidades en cuanto a la elección del tipo de mallado tridimensional: el formado por tetraedros y el formado por prismas rectos de base triangular. La propia naturaleza de los problemas a resolver ha inclinado la elección por el segundo tipo.

Considerando el fluido viento moviéndose de forma envolvente a la superficie del suelo por capas a diferentes alturas, parece más razonable elegir un tipo de elemento prismático recto con base triangular aprovechando la discretización en la malla plana, cuya altura o dimensión vertical sea mucho más pequeña que las dimensiones horizontales, tal que casi se puede considerar como una placa triangular de espesor h.

De hecho, en la realidad se van a estudiar dominios tridimensionales donde la dimensión vertical es mucho más pequeña que las horizontales.

Sin embargo, las caras triangulares del prisma no son horizontales, precisamente para adaptarse a las irregularidades del terreno.

El prisma utilizado es un elemento que tiene cinco caras, dos triangulares y tres cuadriláteras, con un total de seis nodos:



Generador de mallado en 3-D: Construcción de la malla e implementación computacional

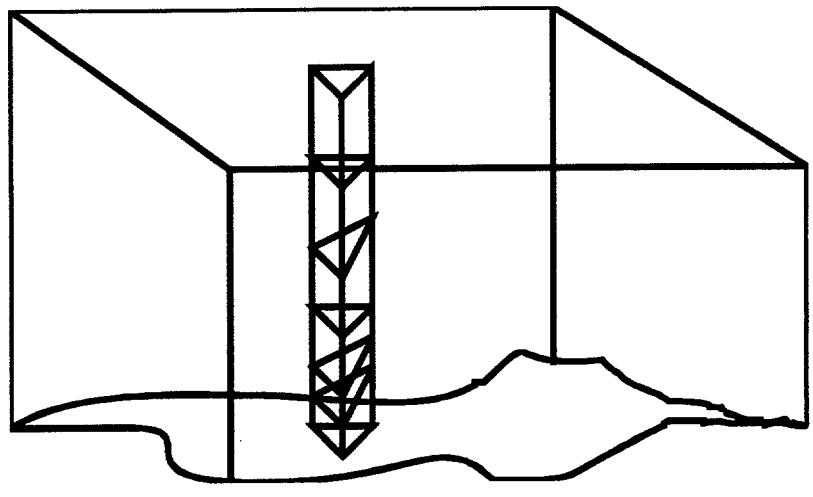
A partir de la malla plana levantada según las cotas dadas por la forma del terreno y conociendo la cota máxima que el usuario impone al dominio tridimensional, se establece el número de capas de elementos prismáticos deseado.

Sobre cada nodo de la malla plana original se van a localizar tantos nodos como capas de elementos se deseen. Las formas de colocar los nodos en la vertical pueden ser diversas, con espaciado uniforme o con espaciado variable con la altura.

Usualmente se necesita mayor precisión a menores alturas, por lo que conviene concentrar más nodos cerca de la superficie del terreno. Si consideramos z_t la cota del terreno y z_m la cota máxima, una ley para generar los n+1 nodos, correspondientes a n capas de elementos, de forma variable con la altura es:

$$z_i = z_t + (z_m - z_t)\frac{i^2}{n^2} \qquad i = 0,1,\ldots,n$$

Obsérvese que $z_0 = z_t$ y $z_n = z_m$. En general, haciendo lo mismo para cada nodo, se puede concluir que se obtienen n nuevas capas de nodos a partir de la inicial, los cuales unidos forman la malla tridimensional de prismas rectos de base triangular.

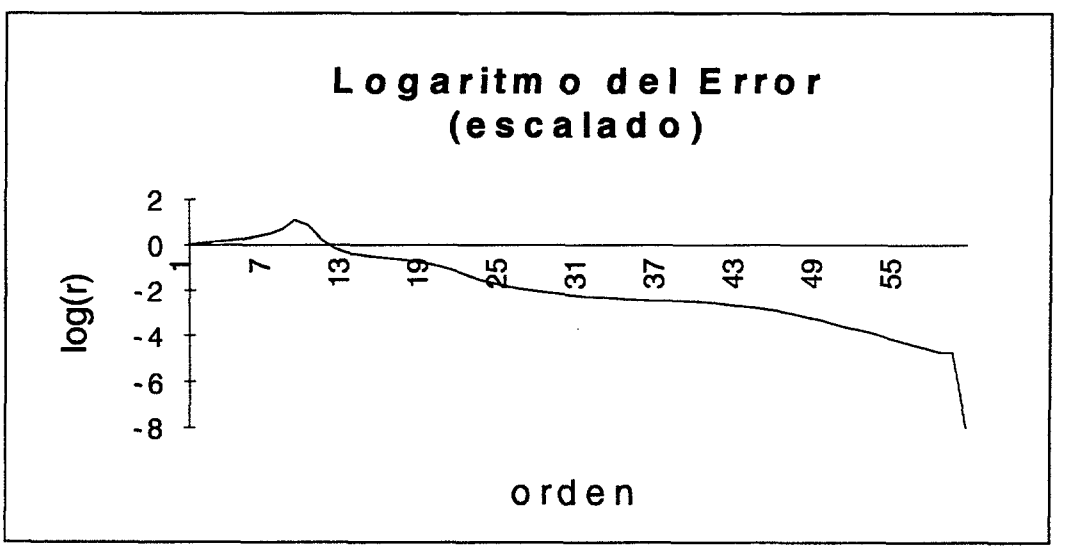


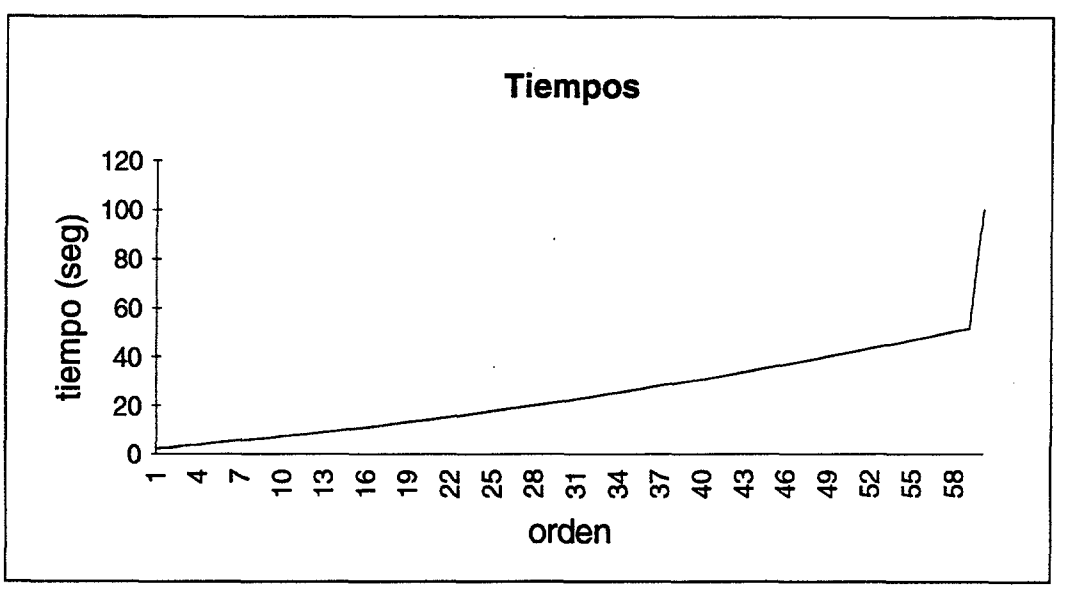
Datos de la resolución del sistema.

Efectuada la resolución del problema considerado mediante FGMRES(v)/SILU(0), para varios casos obtenemos los siguientes datos de resolución; habiendo considerado un mallado plano de base con 2378 nodos, 4560 elementos y 6937 aristas y 10, 15 y 20 capas de altura, imponiéndose una tolerancia de 10^{-7}:

Datos de resolución con λ=1.5 nº de incógnitas=41040

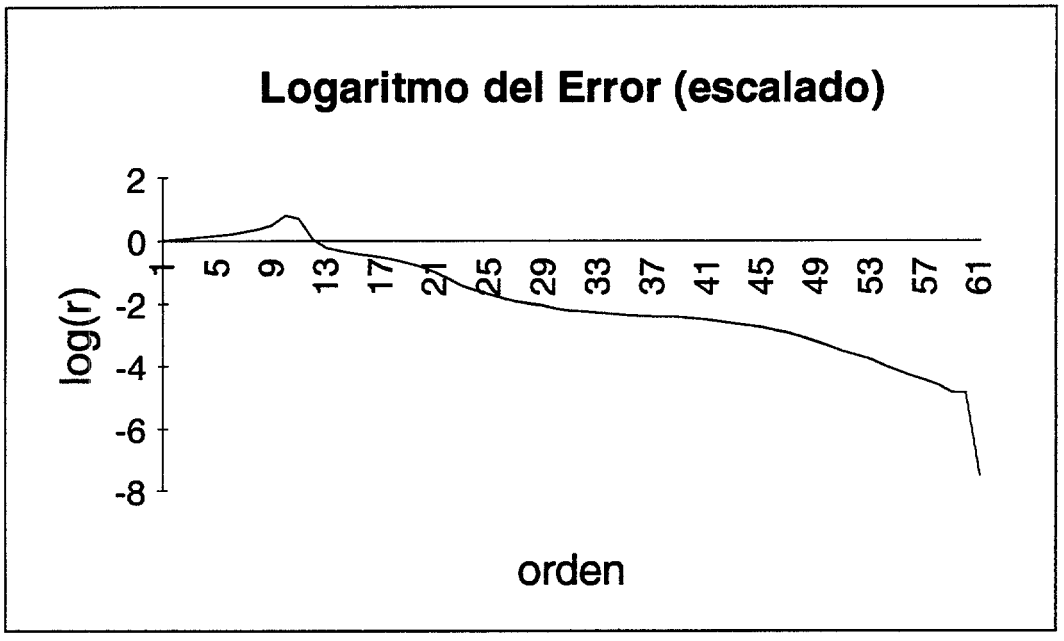
Estación HP9000/730 64 Mb. Memoria RAM

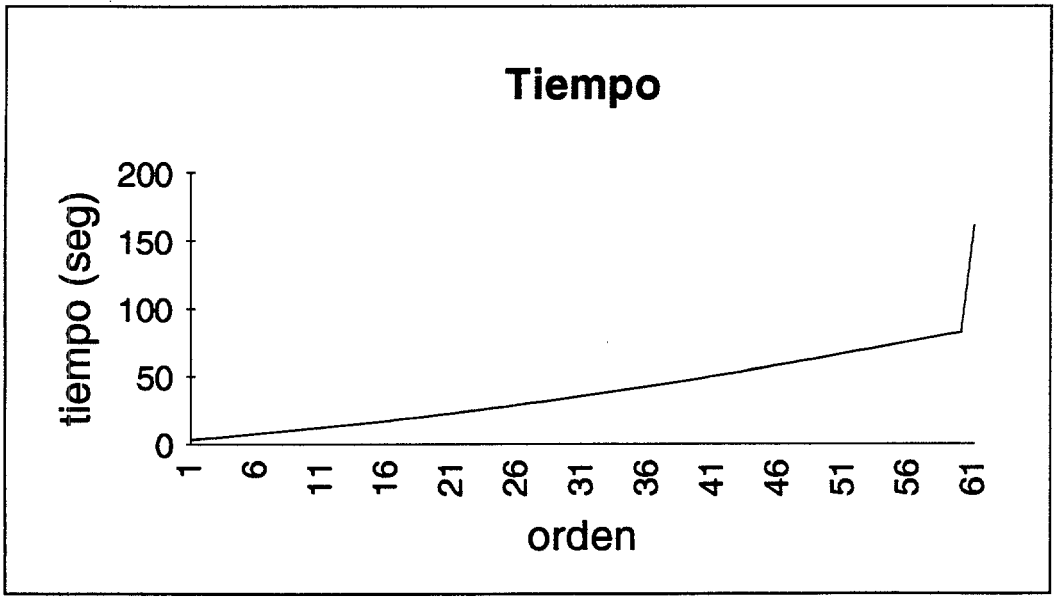




Datos de resolución con λ=1.5 nº de incógnitas=63840

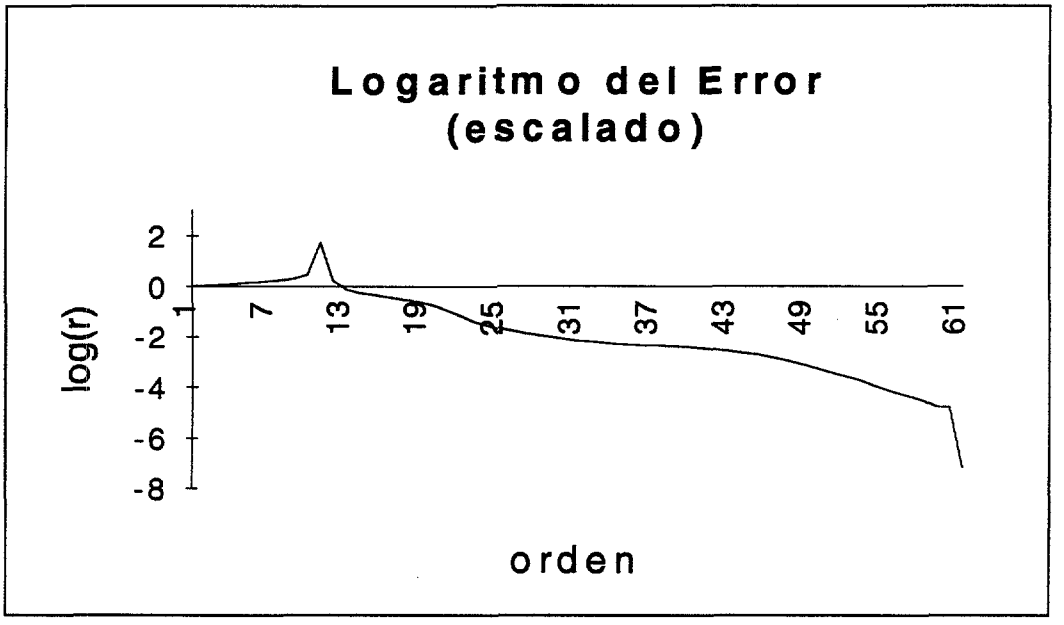
Estación HP9000/730 64 Mb. Memoria RAM

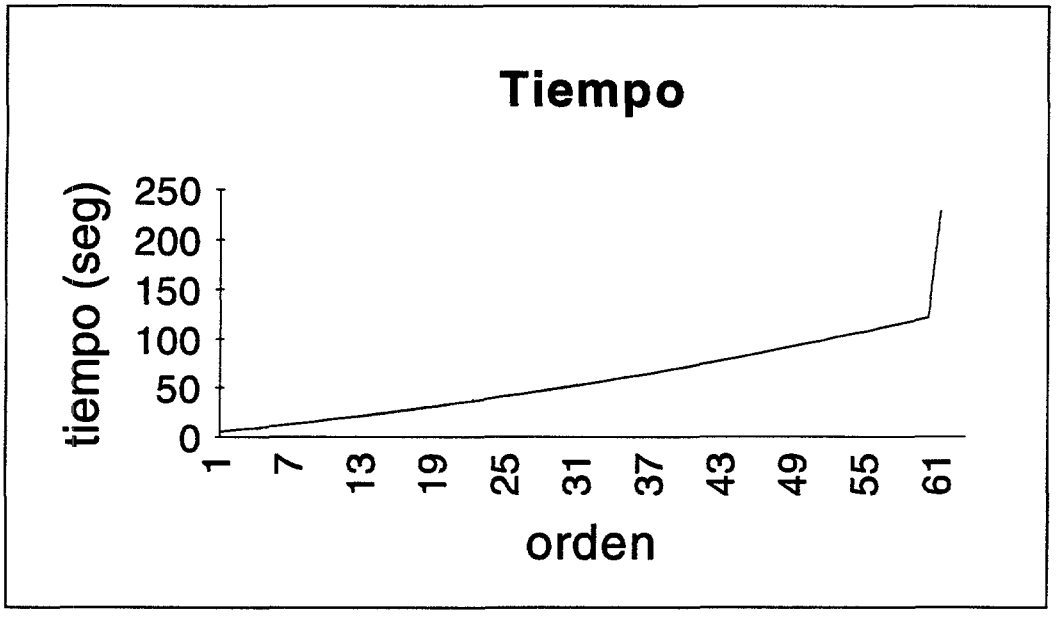




Datos de resolución con λ=1.5 nº de incógnitas=86640

Estación HP9000/730 64 Mb. Memoria RAM





Salida Gráfica

Por último, presentamos los datos de salida por el postprocesador gráfico AVS, donde se representan por escala de colores los datos de módulo de la velocidad del viento en 3D en las zonas de estudio.

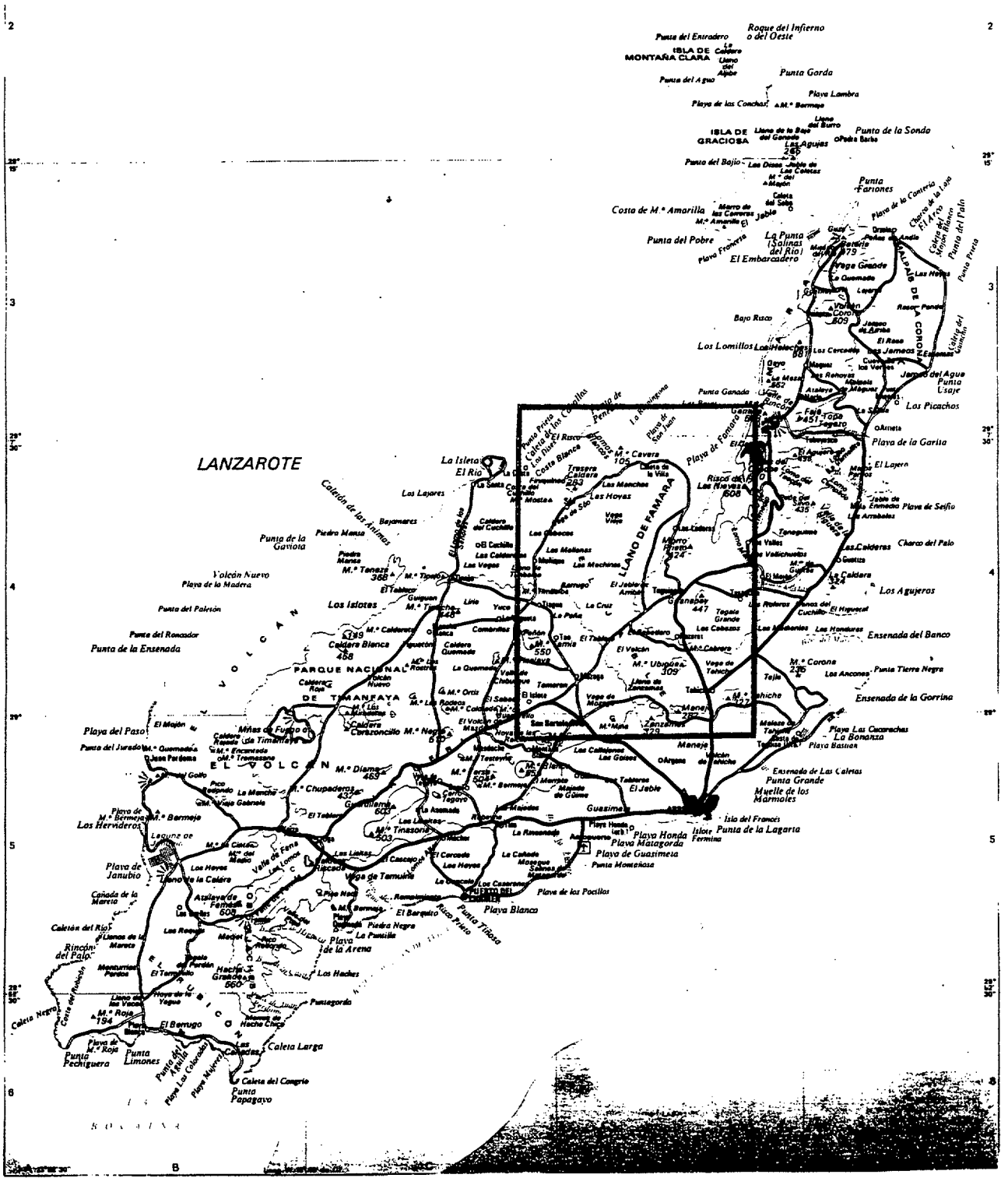


FIGURA VI-1. Zona de estudio en el noreste de la isla de Lanzarote.

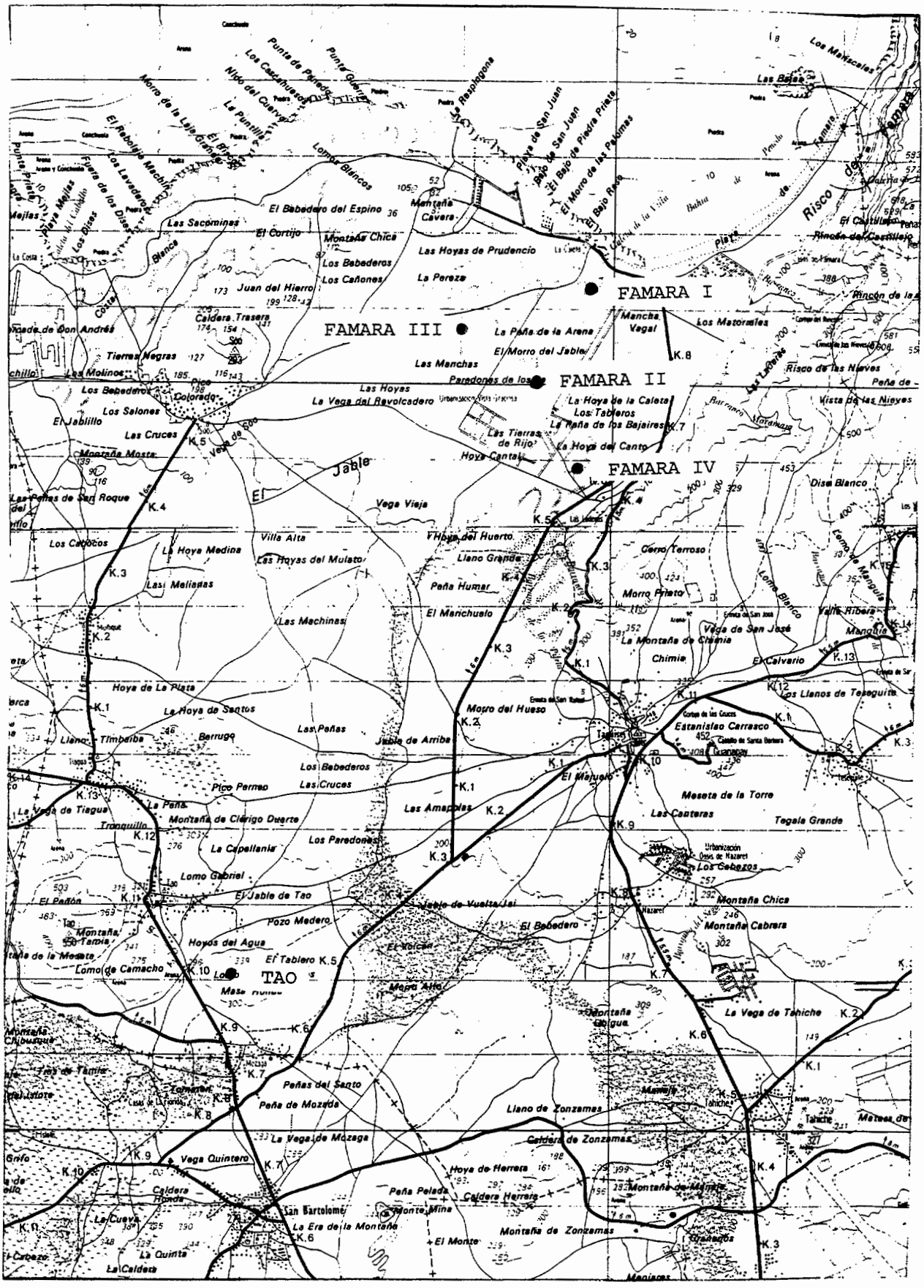
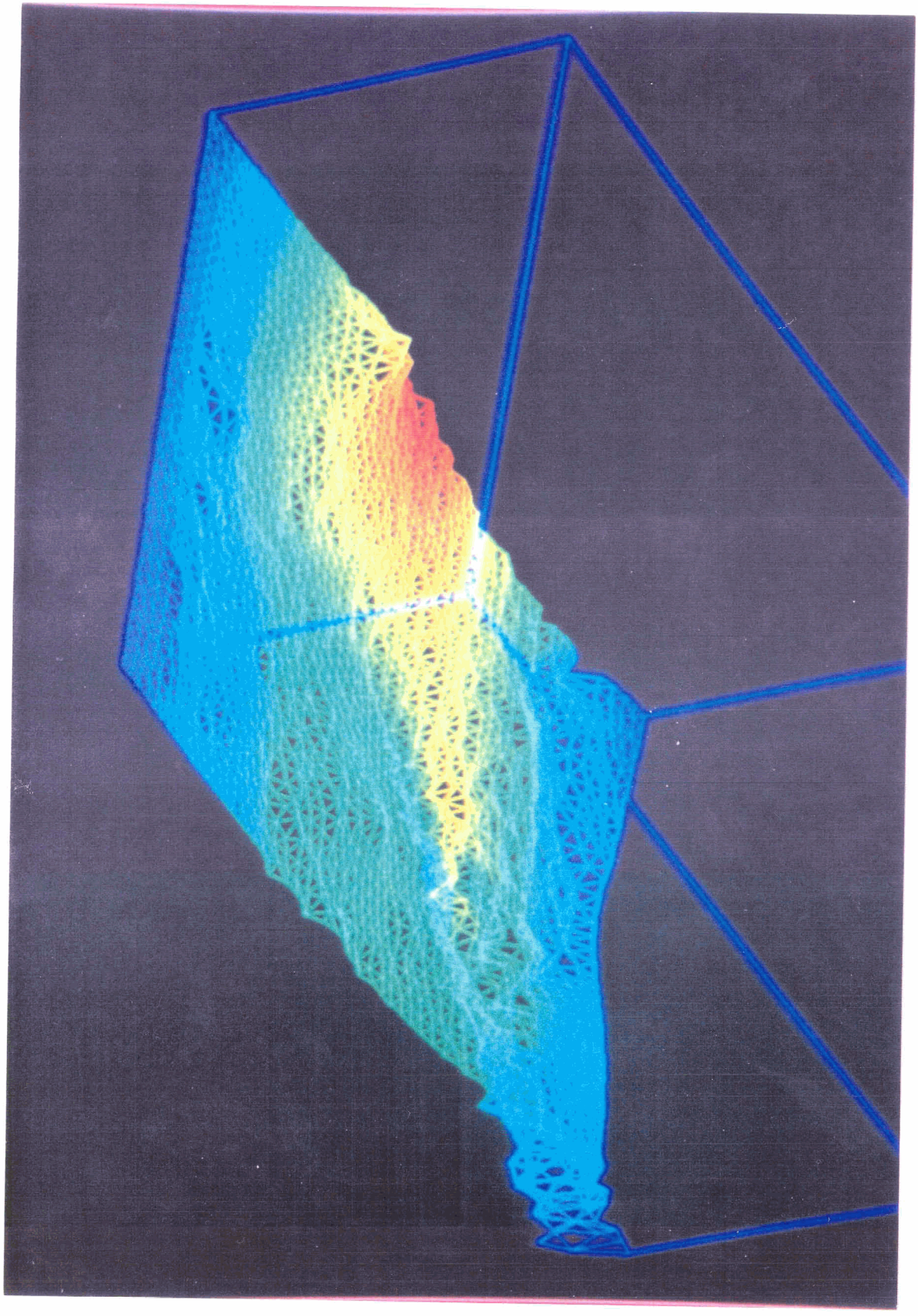
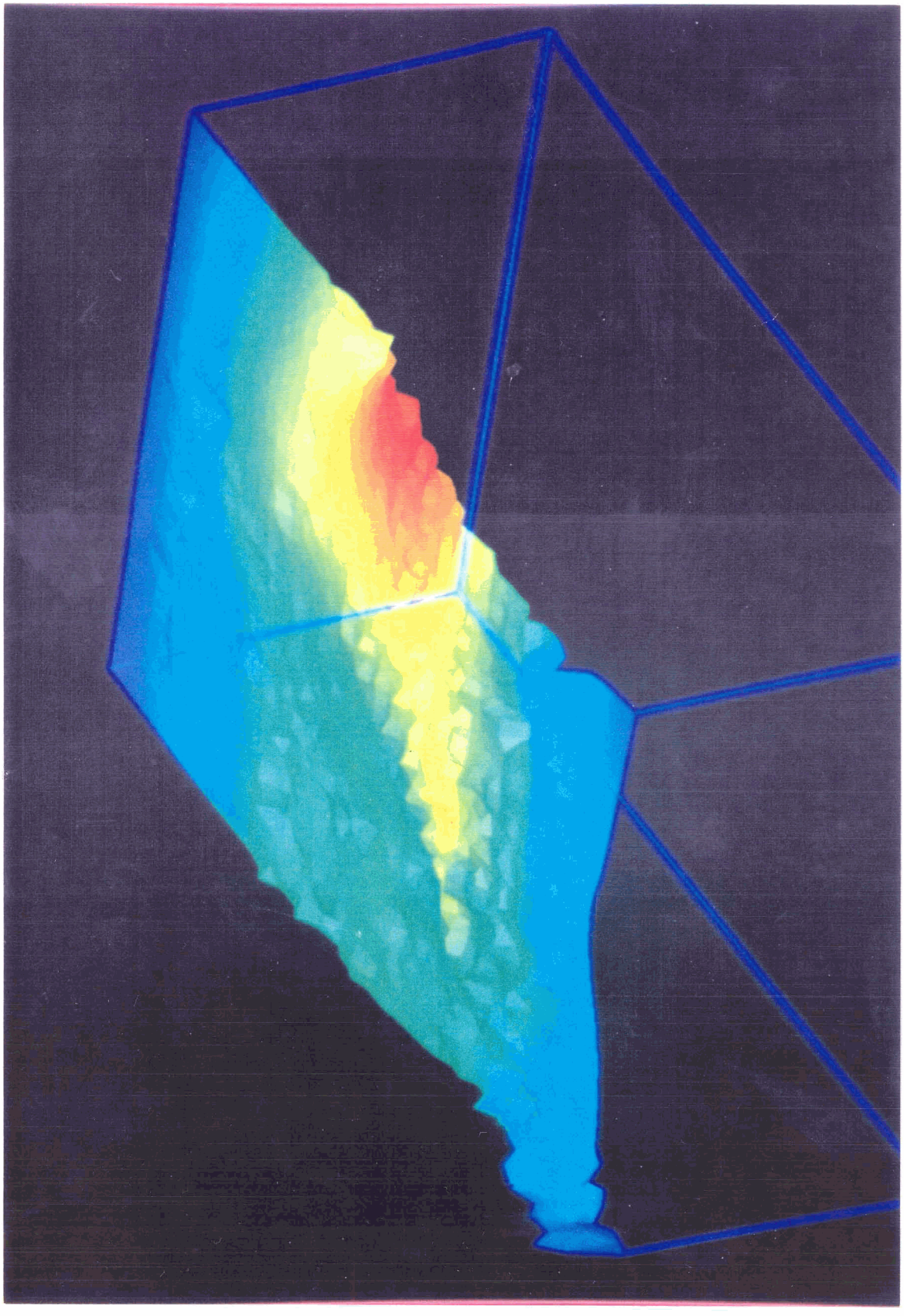


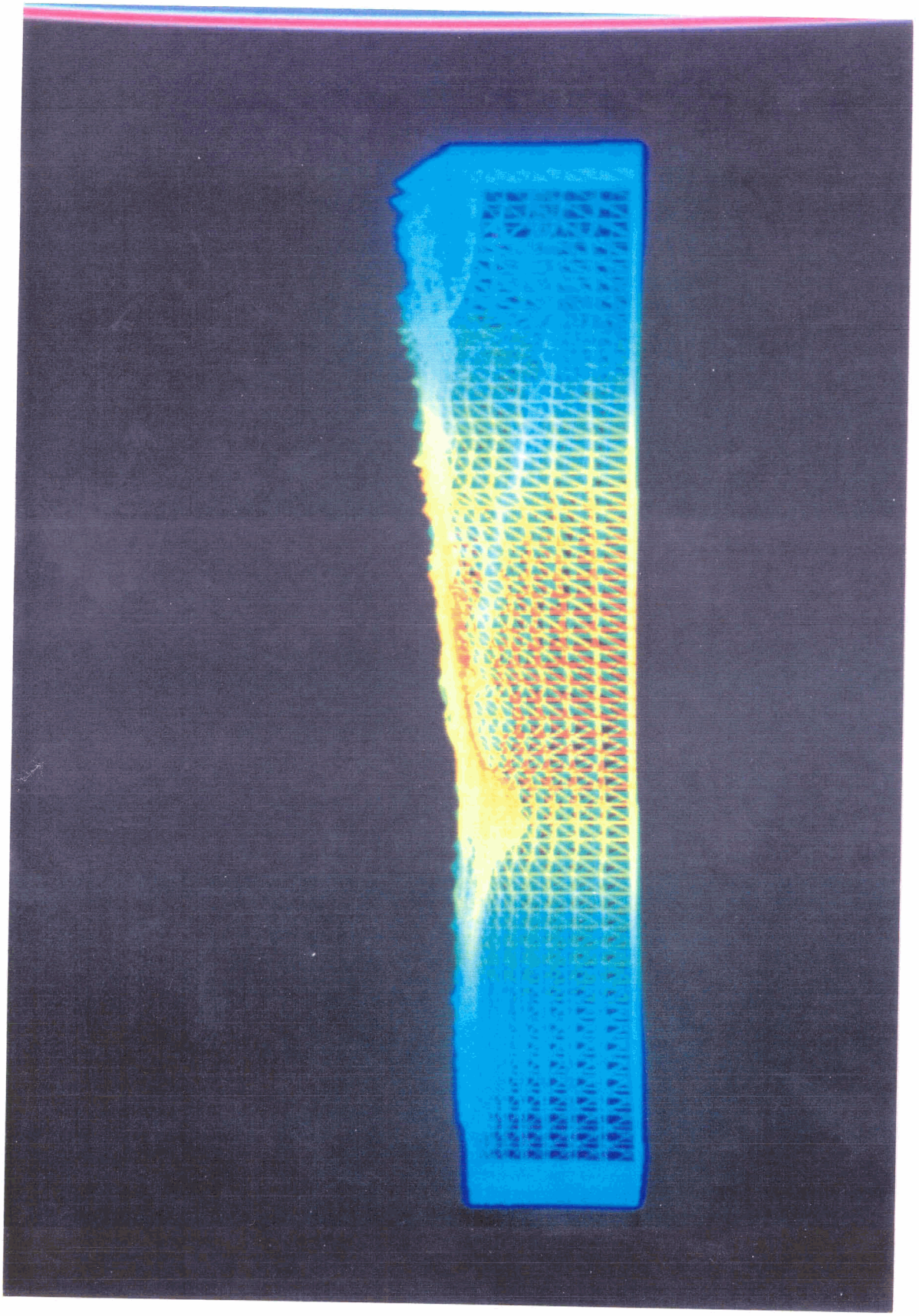
FIGURA VI-2. Zona de estudio de 12x17 Km^2

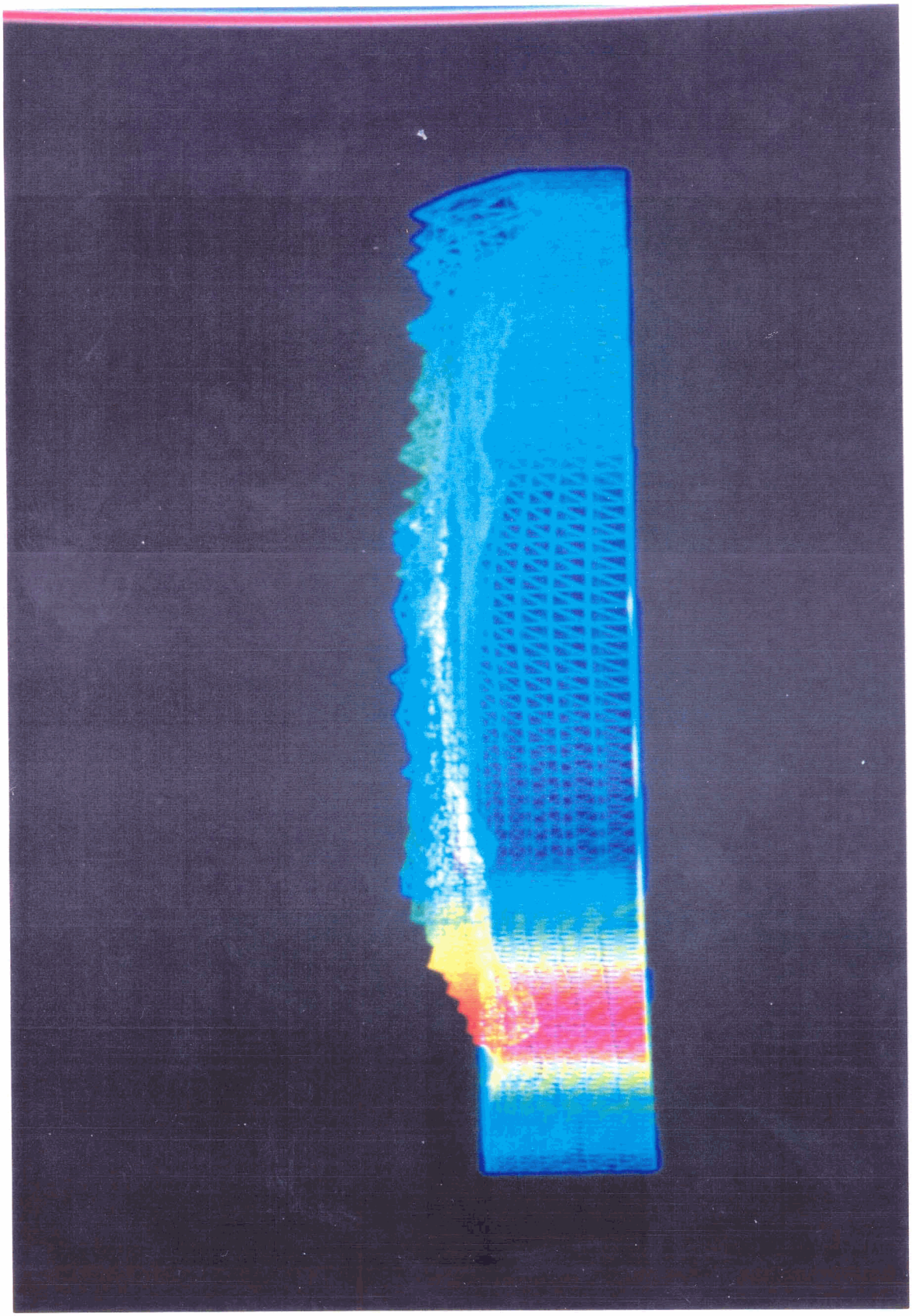
ESTACIÓN	COOR. X (km.)	COOR. Y (km.)	COOR. Z (km.)	DÍA	HORA	|V| (m/s)	DIRECCIÓN (°)	V_X (m/s)	V_Y (m/s)
FAMARA I	7.674	13.245	0.014	20-07-93	12:00	8.71	22.5	-3.33	-8.05
FAMARA II	6.956	12.053	0.044	20-07-93	12:00	9.16	45.0	-6.48	-6.48
FAMARA III	5.973	12.684	0.031	20-07-93	12:00	9.80	19.7	-3.30	-9.23
FAMARA IV	7.583	10.813	0.054	20-07-93	12:00	6.90	7.0	-0.84	-6.85
TAO	3.057	4.097	0.339	20-07-93	12:00	9.80	5.0	-0.85	-9.76

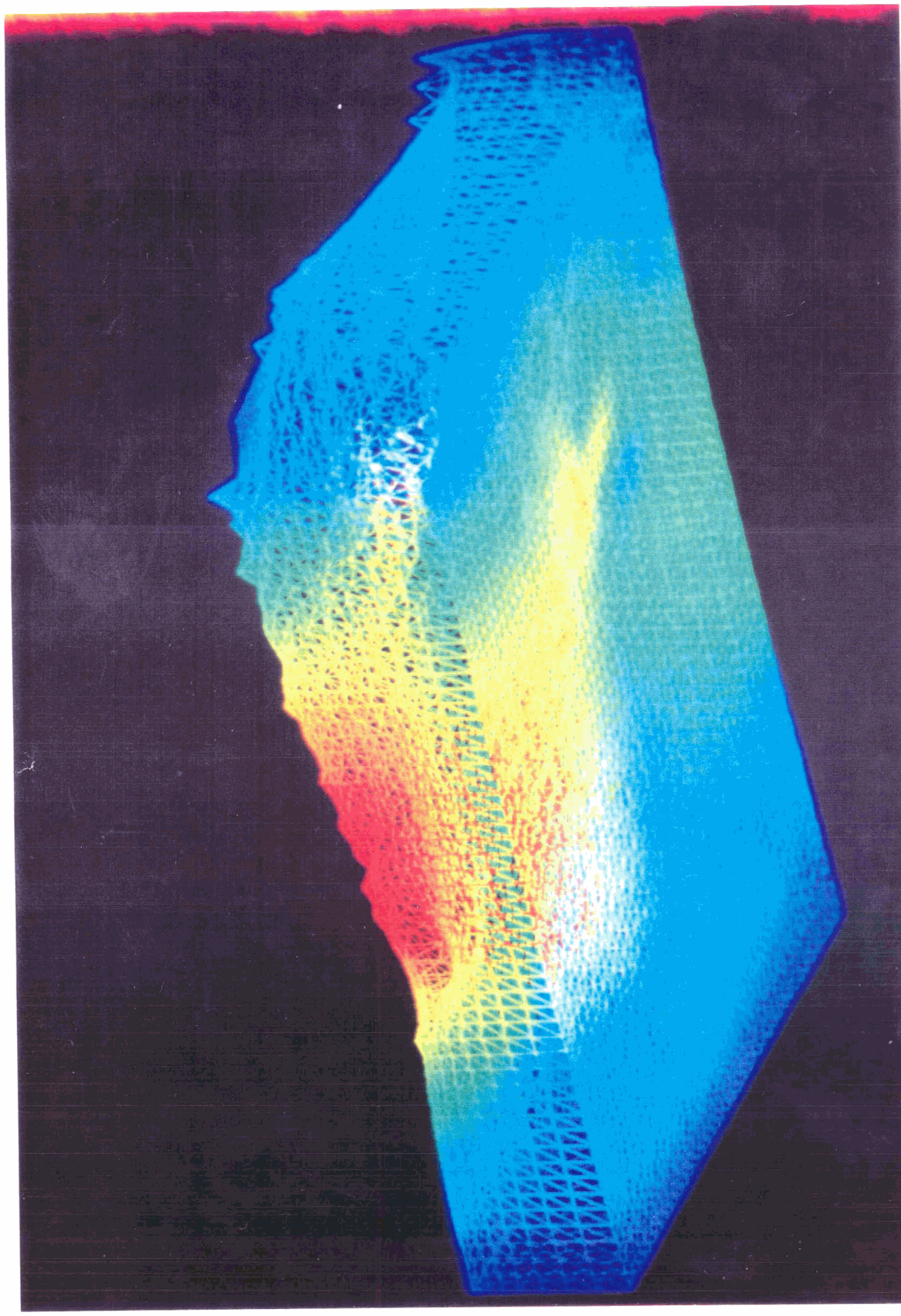
TABLA VI-1. Datos de velocidades de viento.

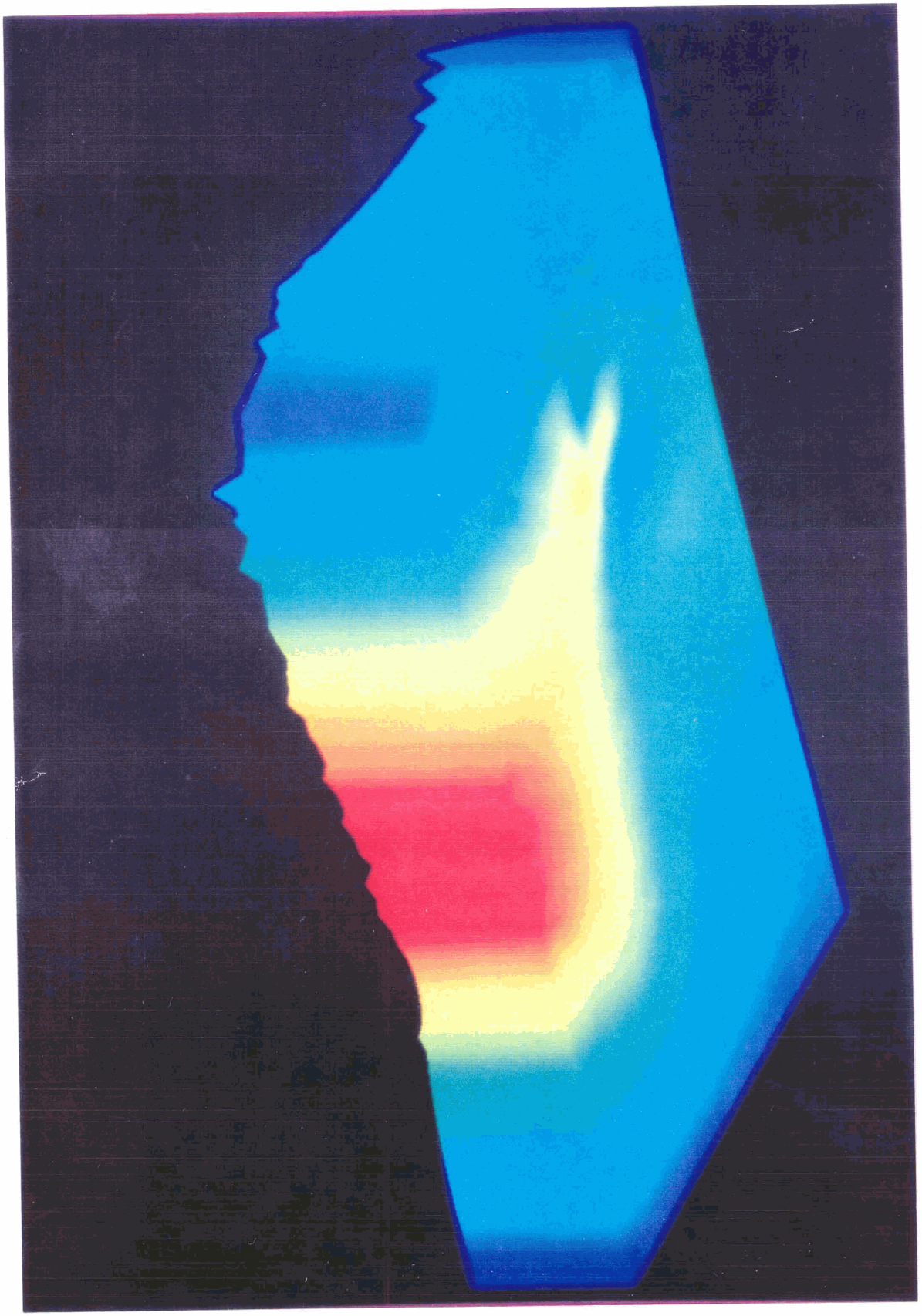


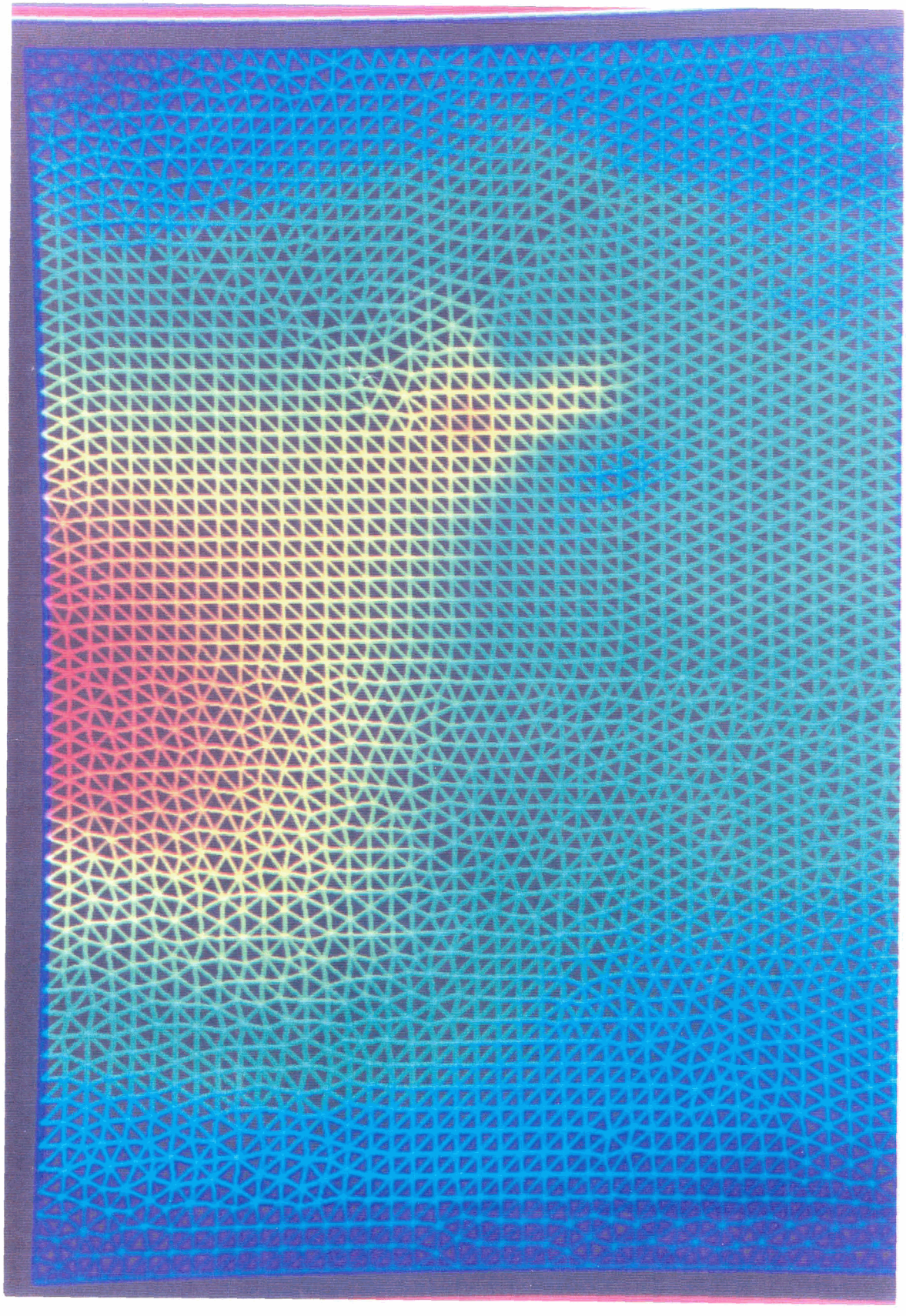


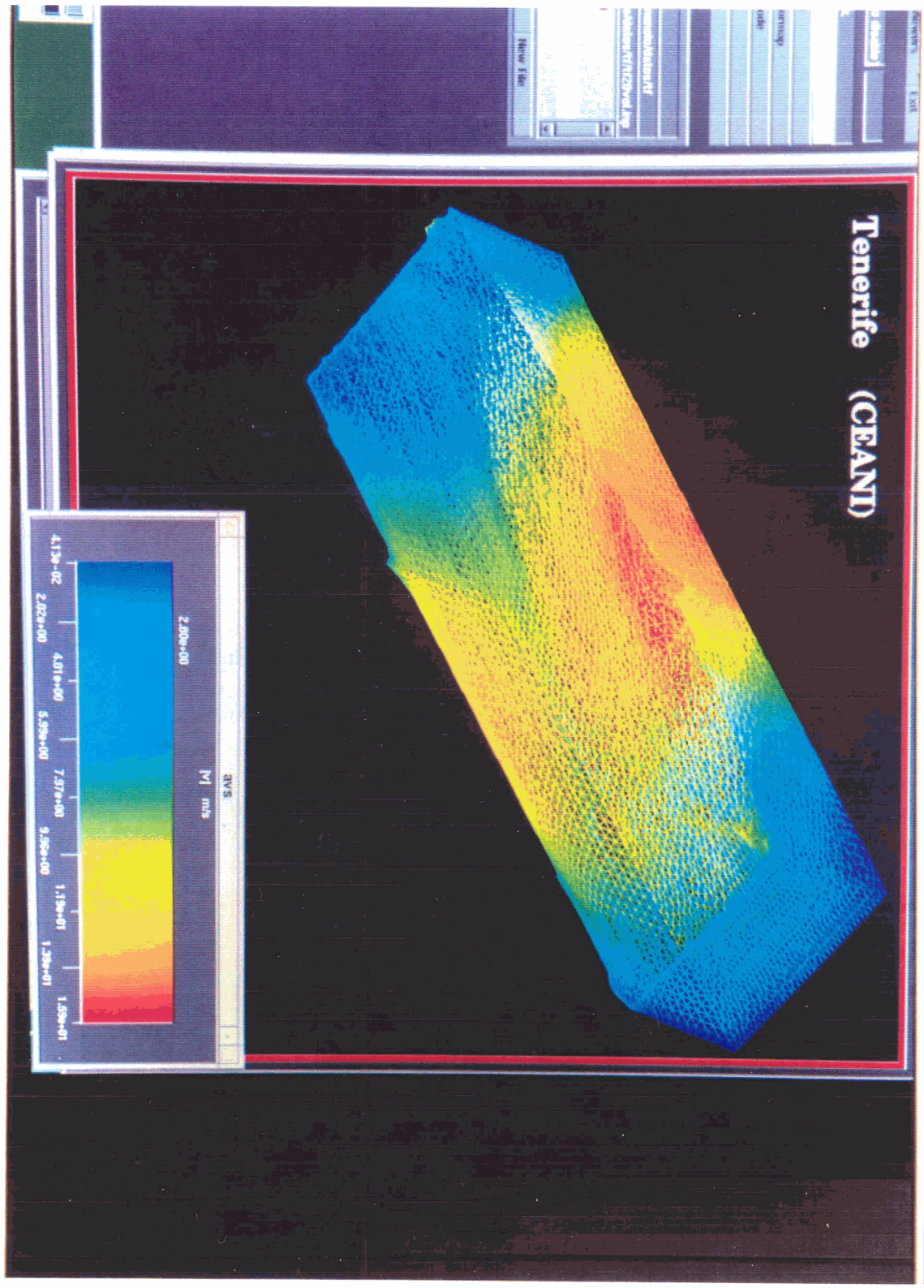












NOTAS FINALES

CONCLUSIONES Y LÍNEAS FUTURAS

Del presente trabajo podemos obtener las siguientes conclusiones:

- Parece conveniente el uso de técnicas iterativas avanzadas en la resolución de grandes sistemas lineales, abandonando los algoritmos clásicos de resolución directa en los casos de sistemas de orden elevado.

- El uso del lenguaje "ANSI-C" conlleva ventajas evidentes en el control y optimización de los recursos del sistema, produce un código fuente altamente transportable y está en rápido proceso de optimización, estandarización y mejora; esto le hace muy apropiado para la investigación científica y el análisis numérico.

- El método de Mínimo Residuo Generalizado (GMRES), puede ser substancialmente mejorado mediante la variabilidad dinámica en la dimensión del espacio de Krylov (tal como se propone en este Trabajo).

- El método de almacenamiento propuesto: "método hipercompacto", conjuga varias ventajas: por un lado minimiza de forma ostensible el uso de la memoria RAM, por otra parte permite el uso de las técnicas de vectorización y paralelización de las operaciones más costosas en la resolución iterativa.

- El método de ensamblaje propuesto, por "cadenas simplemente enlazadas", presenta notables ventajas de flexibilidad, rapidez y optimización de recursos, que le hacen muy apropiado al tratamiento de las matrices "huecas" que aparecen en el Método de los Elementos Finitos.

- No se debe olvidar el gran efecto de aceleración de convergencia que producen los precondicionadores: la técnica de mejora por renumeración parece altamente deseable en este contexto.

- Es interesante el hacer notar que, según se desprende de las aplicaciones estudiadas, las características de convergencia de la matriz dependen, en gran medida del Operador Integral subyacente y se mantienen en las diversas discretizaciones.

- También es destacable el gran efecto que tiene la calidad del mallado en el comportamiento de convergencia de la matriz.

Las posibles líneas futuras que se considerarán para su estudio van en varias direcciones:

- Análisis más profundo de la función de conversión:

    1. Evaluación del perfil de la curva de convergencia con el fin de fijar la dimensión del espacio de Krylov.

    2. Análisis de otras funciones alternativas que relacionen la Tolerancia con la Subtolerancia.

- Mejora del método por aplicación de precondicionadores más potentes y por combinación con otros métodos de resolución iterativa:

3. Precondicionadores iterativos de Richardson y Tchebyshev (ver Golub[23]).

4. Combinación con el BiCGstab de Van der Vorst (GMRESR)

- Aplicación del método a problemas no lineales:

5. Combinación con los algoritmos de Newton-Raphson, Broyden, etc.. de resolución no lineal.

- Resolución por bloques

6. Aplicación del método en la resolución del problema matricial

$$\mathbf{AX = B}$$

Donde se tienen varios sistemas que comparten la matriz "A" (ver Simoncini[45]).

En estos dos últimos casos, sería de especial interés la aplicación de potentes técnicas de precondicionamiento al compartir su coste entre los varios sistemas a resolver

BIBLIOGRAFÍA

REFERENCIAS Y BIBLIOGRAFÍA

[1] J. Abaffy and E. Spedicato, "ABS Projection Algorithms: Mathematical Techniques for Linear and Nonlinear Equations", Ellis Horqood, Chichester, (1989).

[2] P. Almeida. Tesis Doctoral. Departamento de Matemáticas (ULPGC), 1989.

[3] J. Apsimon, and al., "Development of a prototype mesoscale computer model incorporating treatment of topography" - Contract SR 014 UK, (1984).

[4] E. Bodon, "Biconjugate Algorithms in the ABS class I: alternative formulation and theoretical properties", Report DMSIA 3/89, University of Bergamo, (1989).

[5] _____, "Derivation and numerical performance of factorized ABS algorithms for solving linear algebraic equations", Preprint, University of Bergamo, (1990).

[6] E. Bodon and E. Spedicato, "Numerical evaluation of the implicit LU, LQ and QU algorithms in the ABS class", Report DMSIA 28/90, University of Bergamo, (1990).

[7] P. N. Brown, "A theoretical comparison of the Arnoldi and GMRES algorithms", SIAM J.Sci. Statist. Comput., 12, pp. 58-78, (1991).

[8] J. Y. Caneill, P. Racher, R. Rosset. "A finite-element formulation of a variational procedure of wind-field adjustment over complex terrain". Electricite de France, Diection des Estudes et Recherches (1984).

[9] Z. C. Christidis. "Modeling of atmospheric pollutant transport in shorelines", Modern Techniques in Computational Chemistry (MOTECC), 1001-1020, (1990).

[10] P. D. Cuesta Moreno. "Contribución a la discretización de dominios planos no convexos mediante algoritmos de triangulación y técnicas de regularización de mallas". E.T.S.I.I., Tesis Doctoral, Las Palmas de G.C., (1993).

[11] P. D. Cuesta, G. Montero, G. Winter, M. Galán. "Una estrategia de generación y adaptación de mallas en elementos finitos", Centro de Aplicaciones Númericas en Ingeniería (CEANI), U.L.P.G.C., Las Palmas G.C., (1993).

[12] E. H. Cuthill and J.M. McKee, "Reducing the band width of sparse symmetric matrices", Proc. 24th. National Conference of the Association for Computing Machinery, (157-172), (1969).

[13] M. H. Dickerson, "A mass-consistent atmospheric flux models for regions with complex terrain", J. Appl. Met., 17, 241-253 (1978).

[14] J. J. Dongarra, I.S. Duff, D.C. Sorensen and H.A. van der Vorst, "Solving Linear Systems on Vector and Shared Memory Computers", SIAM, Philadelphia, (1991).

[15] L. C. Dutto, "The effect of ordering on Preconditioned GMRES algorithm, for solving the Compressible Navier-Stokes equations", Int. J. Num. Meth. Eng., 36, 457-497, (1993).

[16] L. Ferragut. "NEPTUNO, un sistema adaptativo de elementos finitos", Dpto. de Matemática Aplicada y Métodos Informáticos, E.T.S.I. de Minas, Madrid, (1987).

[17] L. Ferragut, R. Montenegro, G. Montero, G. Winter. "Ajuste de un campo de viento mediante elementos finitos mixtos", Memorias I Congreso Métodos Numéricos en Ingeniería, pp. 90-95, Las Palmas de Gran Canaria (1990).

[18] M. Fortin, R. Glowinski. "Methodes de Lagrangien Aumenté", Dunod, Paris (1982).

[19] M. Galán, G. Montero, G. Winter. "A direct solver for the least square problem arising from GMRES(k)", Communications in Applied Numerical Methods, (1.994).

[20] M. Galán, G. Winter, E. Flórez, P. Cuesta, G. Montero. "Effects of Stochastic Ordering on Preconditioned GMRES Algorithm". Computational Structures Technology, Atenas (Grecia), (1994).

[21] A. George, "Computer implementation of the Finite Element Method", Report Stan CS-71-208, also Ph.D. thesis Departement of Computer Science, Stanford University, Stanford, CA, (1971).

[22] N. E. Gibbs, W. G. Poole, Jr and P. K. Stockmeyer, "An algorithm for reducing the band width and profile of a sparse matrix", SIAM J. Numer. Anal., (13), (236-250) (1976).

[23] G. Golub, R. Underwood and J. H. Wilkinson, "Chebyshev semi-iterative methods and second order Richardson iterative methods", Part I and II, Numer. Math., (1961).

[24] G. Golub and C. F. Van Loan, "MATRIX COMPUTATIONS", The John Hopkins University Press, Baltimore, (1989).

[25] M. H. Gutknecht, "Variants of BICGSTAB for matrices with complex spectrum", prel. rep., IPS, ETH-Zürich, (1991)

[26] O. Hassan, K. Morgan and J. Peraire, "An implicit Finite Element Method for high speed flows", 28th. Aerospace Sciences Meeting, Jan. 8-11. Reno. Nevada. AIAA (90-0402), (1990).

[27] K. Hill, R.E. Turner, "NASA's Atmospheric Variability Experiment (AVE)". Bul. Amer. Met. Soc., 58, 170-172, (1977).

[28] G. Marsaglia, "Comments on the perfect uniform random number generator", Unpublished Notes, Wash. S.U.

[29] G. Marsaglia and Tsang, "A fast, easily implemented method for sampling from decreasing or symmetric unimodal density functions", SIAM J.Sci., (1983).

[30] G. Martin, "Méthodes de préconditionement par factorisation incomplète". Mémoire de Maîtrise, Université Laval, Québec, Canada, (1991).

[31] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller and E. Teller, Journal of Chemical Physics, (21), (1087-1092) , (1953).

[32] R. Montenegro, G. Montero, G. Winter, L. Ferragut. "Aplicación de métodos de elementos finitos adaptativos a problemas de convección-difusión en 2-D". Rev. Int. Met. Num. Cal. y Dis. en Ing., Vol. 5, 4, 535-560, (1989).

[33] N. M. Nachtigal, S. C. Reddy and L.N. Trefethen, "How fast are nonsymmetric iterations", SIAM J. Matrix Anal. Appl., 13, 3, pp. 778-795, (1992).

[34] N. M. Nachtigal, L. Reichel and L.N. Trefethen, "A hybrid GMRES algorithm for nonsymmetric linear systems", SIAM J. Matrix Anal. Appl., 13, 3, pp. 796-825, (1992).

[35] I. Orlanski, "A rational subdivision of scale for atmospheric processes", Bul. Amer. Met. Soc., 56, 527-530, (1975).

[36] R. H. Otten and L. P. van Ginneken, "The Annealing Algorithm", Boston: Kluwer, (1989).

[37] C. C. Paige y M. A. Saunders, "LSQR: An algorithm for sparse linear equations and sparse least squares", ACM Trans. Math. Software, 8, 43-71, (1982)

[38] W. H. Press, S. A. Teukolsky, V. T. Vetterling and B. P. Flannery, "Numerical recipes in C", 2nd Edition, Cambridge University Press, (444-451) , (1992).

[39] G. Radicati de Brozolo and M. Vitaletti, "Sparse matrix-vector product and storage representations on the IBM 3090 with Vector Facility", Report G513-4098, IBM-ECSEC, Rome, (1986).

[40] P. A. Raviart, J.M. Thomas. "A mixed finite element method for second order elliptic problems", Mathematical Aspects of Finite Element Method. Gallini, I., Magenes E., Lecture Notes in Math. 606, Springer Verlag, Berlín (1977).

[41] Y. Saad, "A flexible inner-outer preconditioned GMRES algorithm", SIAM J. Sci. Statist. Comput., 14, 2, 461-469, (1993).

[42] Y. Saad and M.H. Schultz, GMRES: "A generalized minimal residual algorithm for solving nonsymmetric linear systems", SIAM J. Sci. Statist. Comput., 7 pp. 856-869, (1986),.

[43] R. L. Sani, P. M. Gresho, D. R. Tuerpe, R. L. Lee, "The imposition of incompressibility constraints via variational adjustment of velocity fields", Proc. First Intern. Conf. on Numerical Methods in Laminar and Turbulent Flow, Londres, (1978).

[44] C. A. Sherman. "A mass consistent model for wind fields over complex terrain". J. Appl. Meteor. 17, 312-319, (1978).

[45] V. Simoncini and E. Gallopoulos, "An Iterative Method for Nonsymmetric Systems with Multiple Right Hand Sides", CSRD Report No. 1242.rev2, 1994.

[46] P. Sonneveld, "CGS: a fast Lanczos-type solver for nonsymmetric linear systems", SIAM J. Sci. Statis. Comp., 10, 36-52, (1989)

[47] O. G. Sutton., "Micrometeorology: A Study of Physical Processes in the Lowest Layers of the Earth´s Atmosphere". New York: McGraw-Hill, (1953).

[48] W. F. Tinney and J. W. Walker, "Direct solutions of sparse network equations by optimally ordered triangular factorization", Proc. IEEE, 55, 1801-1809, (1967).

[49] H. A. Van der Vorst, "Conjugate Gradient type methods for nonsymmetric linear systems", IMACS, Elsevier Sci. Pub, (1992)

[50] _____, "Bi-CGStab: a fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems", SIAM J. Sci. Statis. Comput, (1993)

[51] C. Vuik, "A comparison of some GMRES-like methods, in Iterative Methods in Linear Algebra", pp. 155-162, Elsevier Science Publishers B.V., North-Holland, (1992).

[52] H. F. Walker, "Implementation of the GMRES method using Householder transformations", SIAM J. Sci. Comput., 9, 1, 152-163 (1988).

[53] K. Wark, C. F. Warner, "Contaminación del Aire", Ed. Limusa, (1992).

[54] G. Winter, R. Montenegro, G. Montero, L. Ferragut. "Modelización numérica de distribución de velocidades de campos de viento en parques eólicos". Memoria final del Proyecto de Investigación Subvencionado por la Consejería de Industria y Energía del Gobierno de Canarias. Las Palmas, (1989).

[55] _____. "Modelización numérica de distribución de velocidades de campos de viento en parques eólicos con utilización de métodos adaptativos de elementos finitos y aplicaciones". Memoria final del Proyecto de Investigación Subvencionado por la Consejería de Industria y Energía del Gobierno de Canarias. Las Palmas, (1989).

[56] G. Winter, G. Montero, P. Cuesta and M. Galán. "Mesh Generation and Adaptive Remeshing by Genetic Algorithms on Transonic Flow Simulation". Second European Computational Fluid Dynamics Conference ECCOMAS '94 Stuttgart, (1994)