APLICACION DEL METODO DE DESCOMPOSICION MATRICIAL Y LA TRANSFORMADA RAPIDA DE FOURIER A LA SOLUCION DE LA ECUACION DE POISSON

OSCAR ROJO J. (*)
RICARDO SOTO M. (**)
LUIS VALDIVIA R. (**)

RESUMEN.

Se presenta un algoritmo para resolver numéricamente la Ecuación de Poisson, basado en el método de "Descomposición Matricial" o "Separación Discreta de Variables" con el uso de la Transformada Rápida de Fourier. El método se describe y analiza para condiciones de borde del tipo Dirichlet en regiones rectangulares. Se dan ejemplos y se presentan los programas computacionales que permiten aplicar el método.

1. INTRODUCCION.

Consideremos la Ecuación de Poisson

(1.1) $-\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) = f(x,y)$ para (x,y) en el interior de R , con las condiciones de borde

(1.2) $u(x,y) = g(x,y)$ para $(x,y) \in \partial R$, donde

(*) Ph.D. in Mathematics.

Profesor Depto. Matemáticas U. del Norte

(**) Magister en Matemáticas.

Profesor Depto. Matemáticas U. del Norte

$R = \{(x,y)/a \leq x \leq b, c \leq y \leq d\}$ y ∂R denota la frontera de R . Supongamos que el dominio R puede ser discretizado definiendo una malla.

$$P = \{(x_i, y_i)/x_i = x_0 + ih, y_i = y_0 + jh, i = 0, 1, \dots, n; j = 0, 1, \dots, m\}$$

y aproximemos la ecuación (1.1) por el operador de diferencias de cinco puntos, lo que nos lleva a obtener

$$(1.3) \quad -V_{i+1,j} - V_{i-1,j} - V_{i,j+1} - V_{i,j-1} + 4V_{ij} = h^2 f_{ij},$$

donde V_{ij} es una aproximación para $u(x_i, y_j)$; $i = 1, 2, \dots, n-1$; $j = 1, 2, \dots, m-1$, con condiciones

$$(1.4) \quad \begin{cases} u_{0j} = g(x_0, y_j) ; & u_{nj} = g(x_n, y_j) \\ u_{i0} = g(x_i, y_0) ; & u_{im} = g(x_i, y_m) \end{cases}$$

La ecuación (1.3) conduce finalmente al sistema lineal

$$(1.5) \quad Mx = b, \quad \text{con}$$

$$M = \begin{bmatrix} A & -I & & & \bigcirc \\ -I & A & -I & & \\ & -I & A & -I & \\ \bigcirc & & & -I & A \end{bmatrix}; \quad A = \begin{bmatrix} 4 & -1 & & & \bigcirc \\ -1 & 4 & -1 & & \\ & -1 & 4 & -1 & \\ \bigcirc & & & -1 & 4 \end{bmatrix}$$

y donde M es una matriz de orden $(n-1)(m-1) \times (n-1)(m-1)$, A de orden $(n-1) \times (n-1)$ e I la matriz identidad de orden $(n-1) \times (n-1)$. X es el vector de incógnitas V_{ij} y b el vector de constantes del segundo miembro de (1.3) que incluye las condiciones de borde (1.4).

El problema ahora, es resolver el sistema lineal (1.5).

Hacia 1945 se tenía un considerable conocimiento teórico respecto de la solución de un problema elíptico. Sin embargo, en contraste con esto, sólo problemas en que raramente el número de incógnitas excedía de 20, eran resueltos en la práctica. Significativos avances en la solución de tales problemas, tuvieron su origen en el advenimiento de avanzados computadores y en la invención de poderosos métodos, llegando a ser rutinario más tarde la solución de grandes sistemas del orden de 10^4 incógnitas.

Los métodos de uso más frecuente para resolver el sistema lineal (1.5) han sido de dos tipos: directos e iterativos. A iguales condiciones de esfuerzo y tiempo computacional, los métodos directos son preferidos puesto que llevan a la solución exacta del problema, salvo errores de redondeo. En los últimos años se han venido usando con éxito métodos directos especiales para matrices esparcidas, los cuales a partir de los trabajos de Hockney han sido complementados con el uso del análisis de Fourier. Bajo esta perspectiva estudiamos el método de Descomposición Matricial y la Transformada Rápida de Fourier.

2. METODO DE DESCOMPOSICION MATRICIAL.

2.1. Consideremos un sistema de ecuaciones lineales como el de (1.5)

$$(2.1) \quad Mx = b,$$

en que la matriz coeficientes M es real y simétrica de orden $pq \times pq$ y tridiagonal en bloques de la forma

$$x_j = \begin{bmatrix} x_{1j} \\ x_{2j} \\ \vdots \\ x_{pj} \end{bmatrix}, \quad b_j = \begin{bmatrix} b_{1j} \\ b_{2j} \\ \vdots \\ b_{pj} \end{bmatrix}$$

Así, x y b son vectores pq -dimensionales.

Ahora, el sistema (2.1) puede ser escrito como

$$(2.2) \quad \begin{cases} Ax_1 + Tx_2 & = b_1 \\ Tx_{j-1} + Ax_j + Tx_{j+1} & = b_j \quad ; j = 2, 3, \dots, q-1 \\ Tx_{q-1} + Tx_q & = b_q \end{cases}$$

En orden a hacer de algún modo eficiente el proceso de resolver el sistema en bloques (2.2), el que puede ser de gran tamaño, definimos

$$(2.3) \quad \begin{cases} \bar{x}_j = Q^T x_j \\ \bar{b}_j = Q^T b_j \quad ; \quad j = 1, 2, \dots, q \end{cases}$$

Entonces, después de premultiplicar por Q^T y sustituir (2.3), el sistema (2.2) llega a ser:

$$(2.4) \quad \begin{cases} A\bar{x}_1 + Q\bar{x}_2 & = \bar{b}_1 \\ Q\bar{x}_{j-1} + A\bar{x}_j + Q\bar{x}_{j+1} & = \bar{b}_j \quad ; j = 2, 3, \dots, q-1 \\ Q\bar{x}_{q-1} + \mu\bar{x}_q & = \bar{b}_q \end{cases}$$

Pongamos $\Lambda = \text{diag} \{ \lambda_1, \lambda_2, \dots, \lambda_p \}$,

$\Omega = \text{diag} \{ \omega_1, \omega_2, \dots, \omega_p \}$ y consideremos ahora, sólo la i -ésima ecuación en cada una de las ecuaciones del sistema en bloques (2.4)

Esto origina el siguiente sistema lineal para cada

$i, 1 \leq i \leq p$

$$(2.5) \begin{cases} \lambda_i \bar{x}_{i1} + \omega_i \bar{x}_{i2} & = \bar{b}_{i1} \\ \omega_i \bar{x}_{i(j-1)} + \lambda_i \bar{x}_{ij} + \omega_i \bar{x}_{i(j+1)} & = \bar{b}_{ij}, j=2,3,\dots,q-1 \\ \omega_i \bar{x}_{i(q-1)} + \lambda_i \bar{x}_{iq} & = \bar{b}_{iq} \end{cases}$$

Tenemos entonces p sistemas de la forma (2.5), cada uno de tamaño $q \times q$. Definimos ahora, la matriz triagonal $q \times q$.

$$\Gamma_i = \begin{bmatrix} \lambda_i & \omega_i & & & \\ \omega_i & \lambda_i & \omega_i & & \\ & \omega_i & \lambda_i & \omega_i & \\ & & \omega_i & \lambda_i & \omega_i \\ & & & \omega_i & \lambda_i \end{bmatrix} \quad \text{y los vectores}$$

$$\tilde{x}_i = \begin{bmatrix} \bar{x}_{i1} \\ \bar{x}_{i2} \\ \vdots \\ \bar{x}_{iq} \end{bmatrix} ; \tilde{b}_i = \begin{bmatrix} \bar{b}_{i1} \\ \bar{b}_{i2} \\ \vdots \\ \bar{b}_{iq} \end{bmatrix} \quad \text{De este modo el sistema (2.5)}$$

puede escribirse en forma compacta como:

$$(2.6) \quad \Gamma_i \tilde{x}_i = \tilde{b}_i \quad \text{para } i = 1, 2, \dots, p$$

Ahora bien, al proceder de este modo, resolver el sistema (2.6) sólo nos lleva a encontrar $\tilde{x}_i$. Sin embargo nosotros queremos encontrar x_j , o al menos $\tilde{x}_j$ para obtener después $x_j = Q\tilde{x}_j$. Esto se consigue fácilmente del modo siguiente: Una vez computados los $\bar{b}_j = Q^{-1}b_j$, $j=1,2,\dots,q$, éstos son almacenados en memoria como vectores columna (fila) conformando así una matriz $p \times q$ ($q \times p$) que llamaremos B,

$$(2.7) \quad B = \begin{bmatrix} \bar{b}_1 & \bar{b}_2 & \dots & \bar{b}_q \end{bmatrix}$$

Observamos además, que las filas (columnas) de esta matriz B son precisamente los vectores $\bar{b}_i$ de los sistemas (2.6) y que las soluciones $\tilde{x}_i$ de estos sistemas pueden ser guardados como vectores filas (columnas) en los espacios que ocupaban los $\bar{b}_i$, puesto que estos no volverán a ser considerados ($\tilde{x}_i$ en donde estaba almacenado $\bar{b}_1$, $\tilde{x}_2$ en la posición de $\bar{b}_2$, etc.). Finalmente, las columnas (filas) de la matriz B, en su última versión, son los vectores $\tilde{x}_j$.

Resulta entonces el siguiente algoritmo:

- 1° Determinar los autovalores y autovectores

de las matrices A y T, es decir, determinar las matrices Q, Λ y Ω .

2° Computar los vectores $\bar{b}_j$ según (2.3).

3° Resolver los sistemas tridiagonales (2.6).

4° Computar $x_j = Q \bar{x}_j$, $j = 1, 2, \dots, q$.

2.2. En el caso particular de la ecuación de Poisson (1.1) con condiciones de borde (1.2) que nos preocupa, las matrices A e I, de órdenes $(n-1) \times (n-1)$; del sistema lineal (1.5) son diagonalizados por

$$(2.8) \quad Q = \sqrt{\frac{2}{n}} \begin{bmatrix} \sin \frac{\pi}{n} & \sin \frac{2\pi}{n} & \dots & \sin \frac{(n-1)\pi}{n} \\ \sin \frac{2\pi}{n} & \sin \frac{4\pi}{n} & \dots & \sin \frac{2(n-1)\pi}{n} \\ \vdots & \vdots & \ddots & \vdots \\ \sin \frac{(n-1)\pi}{n} & \sin \frac{2(n-1)\pi}{n} & \dots & \sin \frac{(n-1)^2\pi}{n} \end{bmatrix}$$

Además, $\Omega = -I$ y los autovalores de A están dados por:

$$(2.9) \quad \lambda_k = 4 - 2 \cos \frac{k\pi}{n}, \quad k = 1, 2, \dots, n-1$$

La matriz coeficientes de los sistemas (2.6) queda entonces:

$$(2.10) \Gamma_i = \begin{bmatrix} \lambda_i & -1 & & & \\ -1 & \lambda_i & -1 & & \\ & -1 & \lambda_i & -1 & \\ & & -1 & \lambda_i & -1 \\ & & & -1 & \lambda_i \end{bmatrix} \quad \begin{array}{l} \text{de orden} \\ (m-1) \times (m-1) \end{array}$$

El algoritmo anterior puede formularse ahora como sigue:

- 1° Computar $\bar{b}_j = Q^T b_j$, $j = 1, 2, \dots, n-1$
- 2° Resolver los sistemas tridiagonales $\Gamma_i \tilde{x}_i = \tilde{b}_i$, $i = 1, 2, \dots, n-1$.
- 3° Computar $x_j = Q \bar{x}_j$, $j = 1, 2, \dots, m-1$

Los pasos 1° y 3° pueden efectuarse muy eficientemente usando la Transformada Rápida de Fourier.

Queremos comentar aquí, que el método de la Descomposición Matricial también puede ser aplicado a la ecuación de Poisson con otras condiciones de borde, como por ejemplo, de Neumann, periódicas y periodicidad en una variable con condiciones de Dirichlet sobre los lados opuestos horizontales del rectángulo. En estos casos, el sistema lineal que resulta de discretizar la ecuación (1.1) es diferente, pero sus matrices A y T de nuevo conmutan. Entonces existe una matriz ortogonal Q que diagonaliza A y T . Esta matriz Q es conocida, como también lo son los autovalores de A y T , siendo Q de forma tal que permite aplicar la Transformada Rápida de Fourier para computar Qc .

3. LA TRANSFORMADA RAPIDA DE FOURIER.

3.1. La Transformada Rápida de Fourier, FFT (Fast Fourier Transform), es un algoritmo para computar la Transformada Discreta de Fourier de un conjunto de números complejos. Fué publicado en 1965 por James W. Cooley y John W. Tukey y hasta antes de su publicación, programas computacionales ocupaban horas en el cómputo de transformadas discretas de Fourier de N puntos. El número de operaciones requeridas era de N^2 multiplicaciones y $N(N-1)$ adiciones. El algoritmo FFT reduce el número de operaciones a $\frac{N}{2} \log_2 N$ multiplicaciones y $\frac{N}{2} \log_2 N$ adiciones cuando $N=2^Y$. Así, si $\frac{N}{2}$ supone mos que el tiempo computacional es proporcional al número de multiplicaciones, entonces la razón entre el tiempo de cómputo de ambos métodos es

$$\frac{\frac{N^2}{2}}{\frac{N}{2} \log_2 N} = \frac{2N}{\log_2 N}, \text{ lo cual para el caso } N = 1.024 = 2^{10},$$

implica que la FFT reduce el tiempo computacional en un factor superior a 200. Esta notable reducción en el tiempo empleado para computar las transformadas discretas de Fourier, produjo un completo cambio de actitud hacia aquello que puede hacerse en un computador usando métodos de Fourier.

Definición.

Sea $g(j)$, $j=0,1,\dots,N-1$, un conjunto de números complejos. Se llama Transformada Discreta de Fourier (DFT) de $\{g(j)\}$ a.

$$(3.1) \quad G(k) = \sum_{j=0}^{N-1} g(j) e^{-\frac{ijk(2\pi)}{N}}, \quad k=0,1,\dots,N-1$$

En lo sucesivo escribiremos $w = e^{-\frac{i2\pi}{N}}$, con lo cual,

$$(3.2) \quad G(k) = \sum_{j=0}^{N-1} g(j) w^{kj}$$

Usando las relaciones de ortogonalidad

$$\sum_{\ell=0}^{N-1} w^{j\ell} w^{-k\ell} = \begin{cases} N & \text{si } j=k \pmod{N} \\ 0 & \text{en otro caso} \end{cases}$$

se puede obtener fácilmente

$$(3.3) \quad g(j) = \frac{1}{N} \sum_{k=0}^{N-1} G(k) w^{-kj}, \text{ que es conocida como la}$$

transformada Discreta Inversa de Fourier. Entonces $G(k)$ y $g(j)$ constituyen un par de transformadas, lo que se de nota por

$$G(k) \longleftrightarrow g(j)$$

De (3.2) y (3.3) respectivamente, podemos ver que $G(k)$ y $g(j)$ son periódicas con periodo N , de modo que podemos considerarlas definidas para todo k y j enteros, con

$$G(k+nN)=G(k) \text{ y } g(j+nN) = g(j), \quad n \text{ entero}$$

Nos interesa ahora describir el algoritmo FFT, para lo cual consideraremos aquí, sólo el caso $N=2^\gamma$. Presentamos primero el caso particular $\gamma = 3$:

Puesto que $0 \leq j, k \leq 7$, podemos escribir

$$j = 4j_2 + 2j_1 + j_0, \quad j_i = 0,1$$

$$k = 4k_2 + 2k_1 + k_0, \quad k_i = 0,1, \text{ con lo cual}$$

(3.2) queda

$$(3.4) \quad G(k_2 k_1 k_0) = \sum_{j_0=0}^1 \sum_{j_1=0}^1 \sum_{j_2=0}^1 g_0(j_2 j_1 j_0) w^{(4k_2+2k_1+k_0)(4j_2+2j_1+j_0)},$$

donde g ha sido llamada g_0 .

Puesto que $w^8 = w^8 = 1$, entonces

$$w^{(4k_2+2k_1+k_0)(4j_2)} = w^{4k_0 j_2} \text{ y } w^{(4k_2+2k_1+k_0)(2j_1)} = w^{(2k_1+k_0)(2j_1)}$$

Luego

$$G(k_2 k_1 k_0) = \sum_{j_0=0}^1 \left[\sum_{j_1=0}^1 \left[\sum_{j_2=0}^1 g_0(j_2 j_1 j_0) w^{4k_0 j_2} \right] w^{(2k_1+k_0)(2j_1)} \right] w^{(4k_2+2k_1+k_0)(j_0)}$$

Finalmente si hacemos

$$g_1(k_0 j_1 j_0) = \sum_{j_2=0}^1 g_0(j_2 j_1 j_0) w^{4k_0 j_2}$$

$$g_2(k_0 k_1 j_0) = \sum_{j_1=0}^1 g_1(k_0 j_1 j_0) w^{(2k_1+k_0)(2j_1)}.$$

(3.5)

$$g_3(k_0 k_1 k_2) = \sum_{j_0=0}^1 g_2(k_0 k_1 j_0) w^{(4k_2+2k_1+k_0)(j_0)}$$

$$G(k_2 k_1 k_0) = g_3(k_0 k_1 k_2),$$

entonces el conjunto de ecuaciones (3.5) describe el algoritmo FFT para $N = 2^Y$ con $\gamma = 3$.

Sea ahora $N=2^\gamma$ para cualquier entero γ . Entonces

$$k = 2^{\gamma-1} k_{\gamma-1} + 2^{\gamma-2} k_{\gamma-2} + \dots + k_0$$

$$j = 2^{\gamma-1} j_{\gamma-1} + 2^{\gamma-2} j_{\gamma-2} + \dots + j_0 \quad \text{y (3.2) queda}$$

$$(3.6) \quad G(k_{\gamma-1} k_{\gamma-2} \dots k_0) = \sum_{j_{\gamma-1}=0}^1 \sum_{j_{\gamma-2}=0}^1 \dots \sum_{j_0=0}^1 g_0(j_{\gamma-1} j_{\gamma-2} \dots j_0) w^p, \text{ donde}$$

$$(3.7) \quad p = (2^{\gamma-1} k_{\gamma-1} + \dots + k_0) (2^{\gamma-1} j_{\gamma-1} + \dots + j_0)$$

Efectuando el producto indicado en (3.7), considerando que $w^N=1$ y efectuando cada suma de (3.6) separadamente se tiene:

$$(3.8) \quad \begin{aligned} g_1(k_0 j_{\gamma-2} \dots j_0) &= \sum_{j_{\gamma-1}=0}^1 g_0(j_{\gamma-1} j_{\gamma-2} \dots j_0) w^{2^{\gamma-1} k_0 j_{\gamma-1}} \\ g_2(k_0 k_1 j_{\gamma-3} \dots j_0) &= \sum_{j_{\gamma-2}=0}^1 g_1(k_0 j_{\gamma-2} \dots j_0) w^{(2k_1+k_0)2^{\gamma-2} j_{\gamma-2}} \\ &\vdots \\ g_\gamma(k_0 k_1 \dots k_{\gamma-1}) &= \sum_{j_0=0}^1 g_{\gamma-1}(k_0 k_1 \dots j_0) w^{(2^{\gamma-1} k_{\gamma-1} + \dots + k_0) j_0} \end{aligned}$$

$$G(k_{\gamma-1} k_{\gamma-2} \dots k_0) = g_\gamma(k_0 k_1 \dots k_{\gamma-1})$$

El conjunto de ecuaciones (3.8) representa la formulación original de Cooley-Tukey para la FFT con $N = 2^Y$.

Ahora bien, usar el conjunto de ecuaciones (3.8) implica efectuar Y sumas, cada uno de los cuales representa N ecuaciones. Cada ecuación a su vez, tiene dos multiplicaciones, una de las cuales es siempre una multiplicación por uno (w^0). Esto requiere entonces NY multiplicaciones. Sin embargo, puesto que

$$w^{p+N/2} = w^p w^{N/2} = -w^p \quad \text{siempre se presenta en el}$$

cómputo del vector $g_\ell(j)$; $1 \leq \ell \leq Y$, el número de multiplicaciones finalmente se reduce a $\frac{NY}{2}$.

3.2 Cómputo de Qc .

Queremos aplicar aquí la Transformada Rápida de Fourier al cómputo del producto $d=Qc$, donde $c = [c_1 \ c_2 \ \dots \ c_{n-1}]^T$ es un vector de componentes reales con $N = 2^{Y+1}$ y Q la matriz ortogonal de la sección 2.2. Entonces tenemos que

$$(3.9) \quad d_k = \sqrt{\frac{2}{n}} \sum_{j=0}^{n-1} c_j \sin \frac{kj\pi}{n}; \quad c_0=0, \quad k=0,1,\dots,N-1, \quad N=\frac{n}{2}$$

Definimos: $p_k = \sqrt{\frac{n}{2}} d_{2k}$

$$q_k = \sqrt{\frac{n}{2}} d_{2k+1} \quad . \quad \text{Entonces}$$

$$(3.10) \quad \left\{ \begin{array}{l} p_k = \sum_{j=0}^{n-1} c_j \operatorname{sen} \frac{2kj\pi}{n} \\ q_k = \sum_{j=0}^{n-1} c_j \operatorname{sen} \frac{(2k+1)j\pi}{n} \end{array} \right.$$

sean

$$C(k) = \sum_{j=0}^{n-1} c_j e^{-\frac{ikj(2\pi)}{n}} \quad y$$

$$\tilde{C}(k) = \sum_{j=0}^{n-1} c_j e^{-\frac{i(2k+1)j\pi}{n}} \quad . \quad \text{Entonces}$$

$$p_k = -\operatorname{Im} C(k) \quad y \quad q_k = -\operatorname{Im} \tilde{C}(k). \quad \text{Además}$$

$$\tilde{C}(k) = \sum_{j=0}^{n-1} \tilde{c}_j e^{-\frac{ikj(2\pi)}{n}}, \quad \text{donde}$$

$$(3.11) \quad \tilde{c}_j = c_j e^{-\frac{ij\pi}{n}}, \quad j = 1, 2, \dots, n-1; \quad \tilde{c}_0 = 0$$

El cómputo de $C(k)$ resulta más eficiente si separamos la sucesión c_j en dos nuevas sucesiones como sigue:

$$(3.12) \quad h_j = c_{2j} ; \quad g_j = c_{2j+1} , \quad j=0,1,\dots,N-1, \quad h_0 = 0$$

Podemos escribir entonces

$$(3.13) \quad \left\{ \begin{array}{l} C(k) = \left[H(k) + e^{\frac{-ikN}{N}} G(k) \right] , \text{ donde} \\ H(k) = \sum_{j=0}^{N-1} h_j e^{\frac{-ikj(2N)}{N}} \\ G(k) = \sum_{j=0}^{N-1} g_j e^{\frac{-ikj(2N)}{N}} \end{array} \right.$$

Computamos simultáneamente las transformadas $H(k)$ y $G(k)$ del modo siguiente:

Formamos la suma $y_j = h_j + ig_j$ y calculamos la FFT de y_j ; $j = 0,1,\dots,N-1$, la que escribimos como:

$$Y(k) = R(k) + iI(k), \text{ donde}$$

$R(k)$ e $I(k)$ son las partes real e imaginaria de $Y(k)$ respectivamente.

Es fácil ver que

$$H(k) = \frac{R(k) + R(N-k)}{2} + i \frac{I(k) - I(N-k)}{2} = R_p + iI_i$$

$$G(k) = \frac{I(k) + I(N-k)}{2} - i \frac{R(k) - R(N-k)}{2} = I_p - iR_i$$

Luego tenemos,

$$(3.14) \quad C(k) = \begin{bmatrix} R_p(k) + I_p(k) \cos \frac{k\pi}{N} - R_i \sin \frac{k\pi}{N} \\ + i \left[I_i(k) - I_p(k) \sin \frac{k\pi}{N} - R_i(k) \cos \frac{k\pi}{N} \right] \end{bmatrix}$$

En consecuencia para $k = 1, 2, \dots, N-1$, se tiene

$$(3.15) \quad d_{2k} = \sqrt{\frac{2}{N}} p_k = \sqrt{\frac{2}{N}} \left[R_i(k) \cos \frac{k\pi}{N} + I_p(k) \sin \frac{k\pi}{N} - I_i(k) \right]$$

Para computar $\tilde{C}(k)$ definimos al igual que en (3.11),

$$(3.16) \quad \tilde{h}_j = \tilde{c}_{2j} ; \quad \tilde{g}_j = \tilde{c}_{2j+1} , \quad j=0, 1, \dots, N-1, \quad \tilde{h}_0 = 0. \text{ Entonces}$$

$$(3.17) \quad \begin{cases} \tilde{H}(k) = \sum_{j=0}^{N-1} \tilde{h}_j e^{\frac{-ikj(2\pi)}{N}} \\ \tilde{G}(k) = \sum_{j=0}^{N-1} \tilde{g}_j e^{\frac{-ikj(2\pi)}{N}} \end{cases} \quad \tilde{C}(k) = \begin{bmatrix} \tilde{H}(k) + e^{\frac{-ik\pi}{N}} \tilde{G}(k) \end{bmatrix} , \text{ donde}$$

Computamos ahora las transformadas de las sucesiones complejas $\tilde{h}_j$ y $\tilde{g}_j$ sucesivamente, esto es,

$$\tilde{H}(k) = R_1(k) + iI_1(k)$$

$$\tilde{G}(k) = R_2(k) + iI_2(k), \text{ donde } R_1 \text{ e } I_1 \text{ representan}$$

las partes real e imaginaria de $\tilde{H}(k)$ respectivamente, mientras que R_2 e I_2 son las correspondientes partes real e imaginaria de $\tilde{G}(k)$. Finalmente tenemos que

$$(3.18) \quad \tilde{C}(k) = \left[R_1(k) + R_2(k) \cos \frac{k\pi}{N} + I_2(k) \sin \frac{k\pi}{N} \right] + \\ + i \left[I_1(k) + I_2(k) \cos \frac{k\pi}{N} - R_2(k) \sin \frac{k\pi}{N} \right]$$

En consecuencia, para $k = 1, 2, \dots, N-1$, obtenemos

$$(3.19) \quad d_{2k+1} = \sqrt{\frac{2}{n}} q_k = \sqrt{\frac{2}{n}} \left[R_2(k) \sin \frac{k\pi}{N} - I_1(k) - I_2(k) \cos \frac{k\pi}{N} \right]$$

En resumen, para computar Q_c tenemos el siguiente algoritmo:

- 1° Efectuar los productos $\tilde{c}_j$ de (3.11)
- 2° Separar las sucesiones c_j y $\tilde{c}_j$ según (3.12) y (3.16)
- 3° Computar mediante el algoritmo FFT las transformadas discretas de $y_j = h_j + i g_j$, $\tilde{h}_j$ y $\tilde{g}_j$.
- 4° Finalmente obtener d_{2k} y d_{2k+1} según (3.15) y (3.19) respectivamente.

4. PROGRAMA COMPUTACIONAL.

El algoritmo de la descomposición matricial para resolver el problema

$$(4.1) \quad -\left(\frac{\alpha^2 \mu}{\alpha x^2} + \frac{\alpha^2 \mu}{\alpha y^2}\right) = f(x,y) \quad , \quad (x,y) \in \text{int } R$$

$$\mu(x,y) = g(x,y) \quad , \quad (x,y) \in \partial R,$$

donde $R = [a, b] \times [c, d]$, ha sido implementado computacionalmente bajo una subrutina denominada DESCOM. Esta subrutina tiene por parámetros los valores de a , b , c y d , el exponente de $n=2^{Y+1}$ bajo el nombre NU, y el valor IS referido a un archivo en la unidad de impresión.

DESCOM llama a las siguientes subrutinas: RHAND, IMPRIM, FFTMOR y TRIDIA.

La subrutina RHAND genera el segundo miembro del sistema de ecuaciones (1.5). Los parámetros de esta subrutina son: la matriz B , en la cual se almacenan por columnas los componentes de dicho segundo miembro; el valor $N=2^{Y+1}-1$; los valores NF y NC definidos como cotas superiores para la dimensión u orden de la matriz B ; los parámetros XA, XB, YC, YD referidos a los vértices del rectángulo R ; y el parámetro H que corresponde a la distancia entre los puntos de la partición del rectángulo. Previamente deben definirse las funciones $F(x,y)$ y $G(x,y)$ correspondientes a $f(x,y)$ y $g(x,y)$ respectivamente.

La subrutina IMPRIM imprime los resultados finales. Según se explicó en la sección 2, estos quedan finalmente almacenados en la matriz B , siendo sus columnas los vectores x_j . El significado de cada parámetro de esta subrutina es claro. El parámetro XREAL es una matriz, $N \times N$ es del orden de ella, IS es un valor ya explicado, y los parámetros F y C corresponden a cotas superiores para la dimensión de la matriz XREAL.

La subrutina FFTMOR tiene por propósito computar los pasos 1º y 3º del algoritmo de la descomposición matricial según el procedimiento expuesto en 3.2. Esta subrutina llama a su vez a la subrutina FFT, correspondiente al algoritmo de la transformada rápida de Fourier. La subrutina FFT que hemos usado es aquella presentada en el texto "The fast Fourier transform", de E. Oran Brigham. El significado de los parámetros de FFTMOR y FFT es suficientemente claro como para no insistir en ello. Sin embargo, conviene recordar que N es una potencia de 2 en nuestro trabajo.

La subrutina TRIDIA resuelve los sistemas tridimensionales

$$\Gamma_i \tilde{x}_i = \tilde{b}_i$$

siguiendo el método que se expone en el apéndice B. Los parámetros de esta subrutina son naturalmente la matriz de coeficientes, su orden y el segundo miembro.

El programa computacional ha sido escrito en lenguaje FORTRAN y sus diferentes subrutinas son presentados en el apéndice 1.

5. EJEMPLOS.

Deseamos presentar ahora resultados numéricos a través de algunos ejemplos, todos ellos referidos a la ecuación (4.1) en el rectángulo $R = [0,1] \times [0,1]$, y ejecutados usando precisión doble.

$$5.1. \quad \begin{cases} f(x,y) = 0 \\ g(x,y) = x + y \end{cases}$$

La solución exacta es claramente $u(x,y) = x + y$. En este caso el error al discretizar la ecuación 1.1 es cero. Por lo tanto, salvo errores de redondeo, la solu-

ción obtenida al emplear cualquier método directo debe ser igual a la solución exacta. Trabajando con $n=8$ obtuvimos los siguientes resultados:

0.2500000	0.3750000	0.5000000	0.6250000	0.7500000
0.8750000	1.0000000			
0.3750000	0.4999999	0.6249999	0.7499999	0.8749999
0.9999999	1.1250000			
0.5000000	0.6249999	0.7499999	0.8749999	0.9999999
1.1249999	1.2500000			
0.6249999	0.7499999	0.8749999	0.9999999	1.1249999
1.2499999	1.3749999			
0.7499999	0.8749999	0.9999998	1.1249998	1.2499998
1.3749998	1.4999999			
0.8749999	0.9999998	1.1249998	1.2499998	1.3749998
1.4999998	1.6249998			
0.9999999	1.1249999	1.2499998	1.3749998	1.4999998
1.6249998	1.7499998			

El valor presentado en la i -ésima fila y j -ésima columna es una aproximación para $\mu(i/8, j/8)$. Fácilmente observamos que efectivamente, salvo errores debido a redondeo, los valores obtenidos corresponden a los valores exactos de $\mu(x,y)$ en tales puntos.

$$5.2 \quad \begin{cases} f(x,y) = \sin x + \sin y \\ g(x,y) = \sin x + \sin y \end{cases}$$

Podemos verificar que $\mu(x,y) = \sin x + \sin y$ es la solución exacta para el problema. Trabajando con $n=8$ obtuvimos las siguientes aproximaciones para $\mu(x,y)$:

0.2493622	0.3721016	0.4909777	0.6041352	0.7098078
0.8063454	0.8922395			
0.3721016	0.4948488	0.6137303	0.7268907	0.8325632
0.9290969	1.0149825			
0.4909777	0.6137303	0.7326154	0.8457775	0.9514495
1.0479804	1.1338600			
0.6041352	0.7268907	0.8457774	0.9589402	1.0646119
1.1611412	1.2470179			
0.7098078	0.8325631	0.9514495	1.0646119	1.1702836
1.2668132	1.3526901			
0.8063453	0.9290968	1.0479802	1.1611411	1.2668131
1.3633449	1.4492262			
0.8922394	1.0149824	1.1338599	1.2470178	1.3526901
1.4492262	1.5351167			

En este caso el error al discretizar la ecuación no es cero y naturalmente no debemos esperar un resultado igual al valor exacto. Consideremos el punto $x = \frac{1}{4}$, $y = \frac{1}{4}$. En tal punto, $\mu(\frac{1}{4}, \frac{1}{4}) = 0.4948077$; en cambio, el valor aproximado obtenido es 0.4948488. Tenemos una aproximación con un error menor que 5×10^{-5} . Esta situación se repite en todos los otros puntos de la partición o malla de R.

APENDICE A. SUBROUTINAS EMPLEADAS EN EL PROGRAMA COMPUTACIONAL.

```

SUBROUTINE DESCOM(XA,XE,YC,YD,NU,IS)
DOUBLE PRECISION PASO(64),B(64,64),PI,DIAGON,H,XA,XB,YC,YD
INTEGER N,NSMALL,FIL,COL
FIL=64
COL=64
N=2*NU
NSMALL=2*(NU+1)
PI=3.141592654
H=(XB-XA)/FLOAT(NSMALL)
CALL RHAND(B,NSMALL-1,FIL,COL,H,XA,XE,YC,YD)

```

C EN B QUEDA EL SEGUNDO MIEMBRO (SE MANEJA EN EL PROGRAMA POR COLUMNAS)

```

DO 30 J=1,NSMALL-1
DO 25 I=1,NSMALL-1
25 PASO(I)=B(I,J)
CALL FFTMOR(PASO,N,NSMALL,NU)
DO 26 I=1,NSMALL-1

```

```

26 B(I,J)=PASO(I)
30 CONTINUE

```

C RESOLUCION DE LOS SISTEMAS TRIDIAGONALES

```

DO 60 L=1,NSMALL-1
DIAGON=4.0-(2.0*DCOS((PI*L)/NSMALL))
DO 44 I=1,NSMALL-1

```

```

44 PASO(I)=B(L,I)
CALL TRIDIA(DIAGON,PASO,NSMALL)

```

C RESULTADO SE DEJA EN B

```

DO 45 I=1,NSMALL-1
45 B(L,I)=PASO(I)
60 CONTINUE

```

C COMPUTO FINAL

```

DO 80 L=1,NSMALL-1
DO 71 I=1,NSMALL-1
71 PASO(I)=B(I,L)
CALL FFTMOR(PASO,N,NSMALL,NU)
DO 72 I=1,NSMALL-1

```

```

72 B(I,L)=PASO(I)*2.0/NSMALL
80 CONTINUE

```

C

```

WRITE(15,88)
CALL IMPPIN(B,NSMALL-1,IS,FIL,COL)
88 FORMAT('0',//40X,'RESULTADOS FINALES',//40X,18('*'),//)
RETURN
END

```

```

SUBROUTINE FFTMOR(C,N,NSMALL,NU)
DOUBLE PRECISION TERM1,TERM2,TERM3,YREAL(64),YIMAG(64)
DOUBLE PRECISION C(NSMALL),D(128)
DOUBLE PRECISION Y2REAL(64),Y2IMAG(64),PI,CIMAG(128)
PI=3.141592654

YREAL(1)=0
YIMAG(1)=C(1)

DO 40 J=2,N
    IPAR=2*J-2
    IMPAR=2*J-1
40    YREAL(J)=C(IPAR)
    YIMAG(J)=C(IMPAR)

CALL FFT(YREAL,YIMAG,N,NU)
DO 70 K=1,N-1
    IPAR=2*K
    TERM1=(DSIN((PI*K)/N))*(YIMAG(K+1)+YIMAG(N-K+1))/2.
    TERM2=(YIMAG(K+1)-YIMAG(N-K+1))/2.
    TERM3=(DCOS((PI*K)/N))*(YREAL(K+1)-YREAL(N-K+1))/2.
70    D(IPAR)=TERM3+TERM1-TERM2

DO 80 J=1,NSMALL-1
    CIMAG(J)=C(J)*DSIN((PI*J)/NSMALL)*(-1.)
80    C(J)=C(J)*DCOS((PI*J)/NSMALL)

YREAL(1)=0
YIMAG(1)=0
Y2REAL(1)=C(1)
Y2IMAG(1)=CIMAG(1)

DO 90 J=2,N
    IPAR=2*J-2
    IMPAR=2*J-1
    YREAL(J)=C(IPAR)
    YIMAG(J)=CIMAG(IPAR)
    Y2REAL(J)=C(IMPAR)
    Y2IMAG(J)=CIMAG(IMPAR)
90    CALL FFT(YREAL,YIMAG,N,NU)
    CALL FFT(Y2REAL,Y2IMAG,N,NU)

DO 100 K=0,N-1
    TERM1=DSIN((PI*K)/N)*Y2REAL(K+1)
    TERM2=DCOS((PI*K)/N)*Y2IMAG(K+1)
    IMPAR=2*K+1
100    D(IMPAR)=TERM1-TERM2-YIMAG(K+1)

DO 110 K=1,NSMALL-1
110    C(K)=D(K)

RETURN
END

```



```

SUBROUTINE FFT(XREAL,XIMAG,N,NU)
DOUBLE PRECISION XREAL(N),XIMAG(N),ARG,P,C,S,TREAL,TIMAG

N2=N/2
NU1=NU-1
K=0
DO 100 L=1,NU
102  DO 101 I=1,N2

        NAUX=K/2**NU1
        P=IBITR(NAUX,NU)
        ARG=6.283185*P/FLOAT(N)
        C=DCOS(ARG)
        S=DSIN(ARG)
        K1=K+1
        K1N2=K1+N2
        TREAL=XREAL(K1N2)*C+XIMAG(K1N2)*S
        TIMAG=XIMAG(K1N2)*C-XREAL(K1N2)*S
        XREAL(K1N2)=XREAL(K1)-TREAL
        XIMAG(K1N2)=XIMAG(K1)-TIMAG
        XREAL(K1)=XREAL(K1)+TREAL
        XIMAG(K1)=XIMAG(K1)+TIMAG
101  K=K+1

        K=K+N2
        IF(K.LT.N)GO TO 102
        K=0
        NU1=NU1-1
        N2=N2/2
100  DO 103 K=1,N

        I=IBITR(K-1,NU)+1
        IF(I.LE.K)GO TO 103
        TREAL=XREAL(K)
        TIMAG=XIMAG(K)
        XREAL(K)=XREAL(I)
        XIMAG(K)=XIMAG(I)
        XREAL(I)=TREAL
        XIMAG(I)=TIMAG
103  CONTINUE
        RETURN
        END

FUNCTION IBITR(J,NU)
J1=J
IBITP=0
DO 200 I=1,NU
        J2=J1/2
        IBITR=IBITR*2+(J1-2*J2)
200  J1=J2

RETURN
END

```

```

SUBROUTINE RHAND(B,N,NF,NC,H,XA,XB,YC,YD)
DOUBLE PRECISION B(NF,NC),H,F,G,XA,YB,YC,YD

DO 10 J=1,N
DO 10 I=1,N
10 B(I,J)=F(XA+I*H,YC+J*H)*(H**2)
DO 20 K=1,N
20 B(K,1)=B(K,1)+G(XA+K*H,YC)
B(K,N)=B(K,N)+G(XA+K*H,YD)
DO 30 K=1,N
30 B(1,K)=B(1,K)+G(XA,YC+K*H)
B(N,K)=B(N,K)+G(XB,YC+K*H)

RETURN
END

```

```

SUBROUTINE XPPIM(XREAL,N,IS,F,C)
INTEGER N,IS,F,C
DOUBLE PRECISION XREAL(F,C)

WRITE(10,*)

DO 10 J=1,N
10 WRITE(10,J+1)(XREAL(J,I)/I=1,N)
RETURN
END

```

```

SUBROUTINE TRIDIA(XLDA,B(N),P(64),Q(64),U(64))
DOUBLE PRECISION XLDA,B(N),P(64),Q(64),U(64)
P(1)=XLDA
Q(1)=1.0/P(1)
U(1)=B(1)/P(1)
DO 10 I=2,N-1
    P(I)=XLDA-w(I-1)
    Q(I)=1.0/P(I)
    U(I)=(B(I)+U(I-1))/P(I)
U(N-1)=U(N-1)
I=N-2
20 IF(I.LT.1)GO TO 30
    P(I)=Q(I)*B(I+1)+U(I)
    I=I-1
    GO TO 20
30 RETURN
END

```

En la programación de estas subrutinas hemos con-
tado con la colaboración del señor Rubén Herrera A., alumno
de la Carrera de Ejecución en Computación e Informática
de la Universidad del Norte.

APENDICE B: UN ALGORITMO PARA RESOLVER SISTEMAS TRIDIAGONALES DE ECUACIONES LINEALES.

Consideremos el sistema de ecuaciones

$$\begin{bmatrix} b_1 & c_1 & & & \\ a_2 & b_2 & c_2 & & \\ & a_3 & b_3 & c_3 & \\ & & \ddots & \ddots & \ddots \\ & & & a_n & b_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_n \end{bmatrix}$$

después de definir $x_0 = x_{n+1} = 0$, este sistema de ecuaciones puede representarse mediante

$$(B.1) \quad a_k x_{k-1} + b_k x_k + c_k x_{k+1} = d_k, \quad k = 1, 2, \dots, n.$$

Definimos

$$(B.2) \quad x_k = u_k x_{k+1} + u_k. \quad \text{Reemplazando en (B.1) tenemos:}$$

$$a_k (u_{k-1} x_k + u_{k-1}) + b_k x_k + c_k x_{k+1} = d_k,$$

de donde podemos escribir

$$(B.3) \quad x_k = \frac{-c_k}{a_k a_{k-1} + b_k} u_{k+1} + \frac{d_k - c_k u_{k-1}}{a_k a_{k-1} + b_k}$$

Comparando (B.2) y (B.3) obtenemos las relaciones

$$q_k = \frac{-c_k}{a_k q_{k-1} + b_k}$$

$$\mu_k = \frac{d_k - a_k \mu_{k-1}}{a_k q_{k-1} + b_k}$$

Si ahora definimos

(B.4) $p_k = a_k q_{k-1} + b_k$, estas relaciones pueden escribirse como:

$$(B.5) \quad q_k = \frac{c_k}{p_k}$$

$$(B.6) \quad \mu_k = \frac{d_k - a_k \mu_{k-1}}{p_k}$$

De $x_1 = -\frac{c_1}{b_1} x_2 + d$, vemos que $q_0 = 0$ y $\mu_0 = 0$.

Haciendo $k=n$ en (B.2), se tiene $x_n = q_n x_{n+1} + \mu_n$. Pero $x_{n+1} = 0$.

Luego, $x_n = \mu_n$.

Hemos deducido así el siguiente algoritmo para resolver el sistema tridiagonal dado:

i) Definir $p_1 = b_1$, $\mu_1 = \frac{d_1}{p_1}$, $q_1 = -\frac{c_1}{p_1}$

ii) Para $k = 2, 3, \dots, n$, computar

p_k , q_k y μ_k por (B.4), (B.5) y (B.6) respectiuvamente.

$$\text{Resulta } x_n = \mu_n$$

iii) Para $k = n-1, n-2, n-3, \dots, 1$, computar x_k por (B.2)

Este algoritmo aplicado al caso en que

$$b_1 = b_2 = b_3 = \dots = b_n = \lambda$$

$$c_1 = c_2 = c_3 = \dots = c_{n-1} = -1$$

$$a_2 = a_3 = \dots = a_n = -1$$

es notablemente simplificado, quedando dado por :

$$\text{i) Definir } p_1 = \lambda, \mu, = \frac{d_1}{p_1}, q_1 = \frac{1}{p_1}$$

ii) Para $k = 2, 3, \dots, n$, computar

$$p_k = \lambda - q_{k-1}$$

$$q_k = \frac{1}{p_k}$$

$$\mu_k = \frac{d_k + \mu_{k-1}}{p_k}$$

Resulta $x_n = \mu_n$.

iii) Para $k = n-1, n-2, n-3, \dots, 1$, computar

$$x_k = q_k x_{k+1} + \mu_k$$

La subrutina TRIDIA ha sido escrita considerando este algoritmo para resolver los sistemas tridiagonales

$$\Gamma_i \tilde{x}_i = \tilde{b}_i, \quad i = 1, 2, \dots, n-1,$$

donde Γ_i es la matriz de (2.10) con $\lambda_i = 4 - 2\cos\frac{i\pi}{n}$

BIBLIOGRAFIA.

- B.L.BUZBEE, G.H.GOLUB, C.W.NIELSON.: "On direct methods for solving Poisson's equations".
Siam Journal on Numerical Analysis.Vol.7
December 1970.
- E. ORAN BRIGHAM. : "The Fast Fourier Transform".
Prentice-Hall, Inc. U.S.A. 1974.
- ALEXANDRA BARBGAS.: "Fast Poisson solvers for problems with sparsity".
Mathematics of Computation, Volume 32,
Number 142. April 1978.
- D. FISCHER, G.GOLUB, O. KALD., G.LLIVA Y O. WIDLUND. "On Fourier-Toeplitz methods for separable elliptic problems". Mathematics of Computation,
Vol. 28, 1974.
- R. W. KOCHNEY. : "A fast direct solution of Poisson's equation using Fourier analysis". J. Assoc.
Computation Machine, Vol. 12. 1965.
- O. WIDLUND. : "On the use of fast methods for separable finite difference equations for the solution of general elliptic problems". Sparse matrices and their applications. Edited by Rose Willoughby Plenum Press, New York, 1972.
- Elliptic problem solvers. Proceeding of the elliptic problem solver conference held in Santa Fe.
New Mexico, June 30 - July 2, 1980. Edited by Martin Schultz. Academic Press.
- WILLIAM AMES. : "Numerical Methods for partial differential equations".
Academic Press - 1977.