

Introducción al Método de Diferencias Finitas y su Implementación Computacional

Antonio Carrillo Ledesma y Omar Mendoza Bernal
Facultad de Ciencias, UNAM
<http://www.mmc.geofisica.unam.mx/acl/>

Una copia de este trabajo se puede descargar de
<http://www.mmc.geofisica.unam.mx/acl/Textos/>

Invierno 2015, Versión 1.0 α

Índice

1	Expansión en Series de Taylor	2
1.1	Aproximación de la Primera Derivada	2
1.1.1	Diferencias Progresivas	2
1.1.2	Diferencias Regresivas	3
1.1.3	Diferencias Centradas	3
1.2	Derivadas de Ordenes Mayores	4
1.2.1	Derivada de Orden Dos	5
1.2.2	Derivadas de Ordenes Mayores	5
1.3	Derivadas en Dos Dimensiones	6
1.4	Derivadas en Tres Dimensiones	7
2	Método de Diferencias Finitas en Una Dimensión	10
2.1	Problema con Condiciones de Frontera Dirichlet	10
2.2	Problema con Condiciones de Frontera Neumann	16
2.3	Problema con Condiciones de Frontera Robin	17
2.4	Discretización del Tiempo	23
2.4.1	Ecuación con Primera Derivada Temporal	23
2.4.2	Ecuación con Segunda Derivada Temporal	27
2.5	Consistencia, Estabilidad, Convergencia y Error del Método de Diferencias Finitas	30
3	Consideraciones Sobre la Implementación de Métodos de Solu- ción de Grandes Sistemas de Ecuaciones Lineales	33
3.1	Métodos Directos	33
3.1.1	Factorización LU	33
3.1.2	Factorización Cholesky	35
3.1.3	Factorización LU para Matrices Tridiagonales	35
3.2	Métodos Iterativos	36
3.2.1	Método de Gradiente Conjugado	38
3.2.2	Método Residual Mínimo Generalizado	40
3.3	Estructura Óptima de las Matrices en su Implementación Com- putacional	41
3.3.1	Matrices Bandadas	43
3.3.2	Matrices Dispersas	44
3.3.3	Multiplicación Matriz-Vector	45
4	Implementación Computacional del Método de Diferencias Fini- tas para la Resolución de Ecuaciones Diferenciales Parciales	47
4.1	Implementación en SciLab	47
4.2	Implementación en Octave (MatLab)	49
4.3	Implementación en C++	51
4.4	Implementación en Python	53
5	Bibliografía	56

1 Expansión en Series de Taylor

Sea $f(x)$ una función definida en (a, b) que tiene hasta la k -ésima derivada, entonces la expansión de $f(x)$ usando series de Taylor alrededor del punto x_i contenido en el intervalo (a, b) será

$$f(x) = f(x_i) + \frac{(x - x_i)}{1!} \left. \frac{df}{dx} \right|_{x_i} + \frac{(x - x_i)^2}{2!} \left. \frac{d^2 f}{dx^2} \right|_{x_i} + \dots + \frac{(x - x_i)^k}{k!} \left. \frac{d^k f}{dx^k} \right|_{\varepsilon} \quad (1.1)$$

donde $\varepsilon = x_i + \theta(x - x_i)$ y $0 < \theta < 1$.

1.1 Aproximación de la Primera Derivada

Existen distintas formas de generar la aproximación a la primera derivada, nos interesa, una que nos de la mejor precisión posible con el menor esfuerzo computacional.

1.1.1 Diferencias Progresivas

Considerando la Ec.(1.1) con $k = 2$ y $x = x_i + \Delta x$, tenemos

$$f(x_i + \Delta x) = f(x_i) + \Delta x \left. \frac{df}{dx} \right|_{x_i} + \frac{\Delta x^2}{2!} \left. \frac{d^2 f}{dx^2} \right|_{\varepsilon_p} \quad (1.2)$$

de esta ecuación obtenemos la siguiente expresión para aproximación de la primera derivada

$$\left. \frac{df}{dx} \right|_{x_i} = \frac{f(x_i + \Delta x) - f(x_i)}{\Delta x} - \frac{\Delta x^2}{2!} \left. \frac{d^2 f}{dx^2} \right|_{\varepsilon_p} \quad (1.3)$$

en este caso la aproximación de $f'(x)$ mediante diferencias progresivas es de primer orden, o sea $O(\Delta x)$. Siendo $O_p(\Delta x)$ el error local de truncamiento, definido como

$$O_p(\Delta x) = -\frac{\Delta x^2}{2!} \left. \frac{d^2 f}{dx^2} \right|_{\varepsilon_p}. \quad (1.4)$$

Es común escribir la anterior expresión como

$$\left. \frac{df}{dx} \right|_{x_i} = \frac{f(x_i + \Delta x) - f(x_i)}{\Delta x} - O_p(\Delta x) \quad (1.5)$$

como

$$f'(x_i) = \frac{f_{i+1} - f_i}{\Delta x} \quad (1.6)$$

para simplificar notación.

1.1.2 Diferencias Regresivas

Considerando la Ec.(1.1) con $k = 2$ y $x = x_i - \Delta x$, tenemos

$$f(x_i - \Delta x) = f(x_i) - \Delta x \left. \frac{df}{dx} \right|_{x_i} + \frac{\Delta x^2}{2!} \left. \frac{d^2 f}{dx^2} \right|_{\varepsilon_r} \quad (1.7)$$

de esta ecuación obtenemos la siguiente expresión para aproximación de la primera derivada

$$\left. \frac{df}{dx} \right|_{x_i} = \frac{f(x_i) - f(x_i - \Delta x)}{\Delta x} - \frac{\Delta x^2}{2!} \left. \frac{d^2 f}{dx^2} \right|_{\varepsilon_r} \quad (1.8)$$

en este caso la aproximación de $f'(x)$ mediante diferencias regresivas es de primer orden, o sea $O(\Delta x)$. Siendo $O_r(\Delta x)$ el error local de truncamiento, definido como

$$O_r(\Delta x) = \frac{\Delta x^2}{2!} \left. \frac{d^2 f}{dx^2} \right|_{\varepsilon_r}. \quad (1.9)$$

Es común escribir la anterior expresión como

$$\left. \frac{df}{dx} \right|_{x_i} = \frac{f(x_i) - f(x_i - \Delta x)}{\Delta x} + O_r(\Delta x) \quad (1.10)$$

como

$$f'(x_i) = \frac{f_i - f_{i-1}}{\Delta x} \quad (1.11)$$

para simplificar notación.

1.1.3 Diferencias Centradas

Considerando la Ec.(1.1) con $k = 3$ y escribiendo $f(x)$ en $x = x_i + \Delta x$ y $x = x_i - \Delta x$, tenemos

$$f(x_i + \Delta x) = f(x_i) + \Delta x \left. \frac{df}{dx} \right|_{x_i} + \frac{\Delta x^2}{2!} \left. \frac{d^2 f}{dx^2} \right|_{x_i} + \frac{\Delta x^3}{3!} \left. \frac{d^3 f}{dx^3} \right|_{\varepsilon_p} \quad (1.12)$$

y

$$f(x_i - \Delta x) = f(x_i) - \Delta x \left. \frac{df}{dx} \right|_{x_i} + \frac{\Delta x^2}{2!} \left. \frac{d^2 f}{dx^2} \right|_{x_i} - \frac{\Delta x^3}{3!} \left. \frac{d^3 f}{dx^3} \right|_{\varepsilon_r} \quad (1.13)$$

restando la Ec.(1.12) de la Ec.(1.13), se tiene

$$f(x_i + \Delta x) - f(x_i - \Delta x) = 2\Delta x \left. \frac{df}{dx} \right|_{x_i} + \frac{\Delta x^3}{3!} \left[\left. \frac{d^3 f}{dx^3} \right|_{\varepsilon_p} + \left. \frac{d^3 f}{dx^3} \right|_{\varepsilon_r} \right] \quad (1.14)$$

esta última expresión lleva a la siguiente aproximación de la primera derivada mediante diferencias centradas

$$\left. \frac{df}{dx} \right|_{x_i} = \frac{f(x_i + \Delta x) - f(x_i - \Delta x)}{2\Delta x} + O_c(\Delta x^2) \quad (1.15)$$

con un error local de truncamiento de segundo orden $O_c(\Delta x^2)$, es decir

$$O_c(\Delta x^2) = \frac{\Delta x^2}{3!} \left[\left. \frac{d^3 f}{dx^3} \right|_{\varepsilon_p} + \left. \frac{d^3 f}{dx^3} \right|_{\varepsilon_r} \right] \quad (1.16)$$

comparado el error local de truncamiento de la aproximación anterior $O_c(\Delta x^2)$, con los obtenidos previamente para diferencias progresivas y regresivas $O_p(\Delta x)$ y $O_r(\Delta x)$, se tiene que

$$\lim_{\Delta x \rightarrow 0} O_c(\Delta x^2) < \lim_{\Delta x \rightarrow 0} O_p(\Delta x). \quad (1.17)$$

Es común encontrar expresada la derivada¹

$$\left. \frac{df}{dx} \right|_{x_i} = \frac{f(x_i + \Delta x) - f(x_i - \Delta x)}{2\Delta x} \quad (1.18)$$

como

$$f'(x_i) = \frac{f_{i+1} - f_{i-1}}{2\Delta x} \quad (1.19)$$

para simplificar notación.

Derivadas Usando Más Puntos Utilizando el valor de la función en más puntos se construyen fórmulas más precisas para las derivadas², algunos ejemplos son

$$\begin{aligned} f'(x_i) &= \frac{-3f_i + 4f_{i+1} - f_{i+2}}{2\Delta x} + O(\Delta x^2) \\ f'(x_i) &= \frac{3f_i - 4f_{i-1} + f_{i-2}}{2\Delta x} + O(\Delta x^2) \\ f'(x_i) &= \frac{2f_{i+1} + 3f_i - 6f_{i-1} + f_{i-2}}{6\Delta x} + O(\Delta x^3) \\ f'(x_i) &= \frac{f_{i-2} - 8f_{i-1} + 8f_{i+1} - f_{i+2}}{12\Delta x} + O(\Delta x^4) \\ f'(x_i) &= \frac{-25f_i + 48f_{i+1} - 36f_{i+2} + 16f_{i+3} - 3f_{i+4}}{12\Delta x} + O(\Delta x^4) \end{aligned} \quad (1.20)$$

1.2 Derivadas de Ordenes Mayores

De forma análoga se construyen aproximaciones en diferencias finitas de orden mayor, aquí desarrollaremos la forma de calcular la derivada de orden dos en diferencias centradas.

¹En el caso de que la derivada sea usada en una malla no homogénea, es necesario incluir en la derivada a que Δx se refiere, por ejemplo en cada punto i , tenemos la Δx_{i-} (por la izquierda) y la Δx_{i+} (por la derecha), i.e. $\left. \frac{df}{dx} \right|_{x_i} = \frac{f(x_i + \Delta x_{i-}) - f(x_i - \Delta x_{i+})}{(\Delta x_{i-}) + (\Delta x_{i+})}$.

²Al usar estas derivadas en el método de diferencias finitas mostrado en la sección (2) las matrices generadas no serán tridiagonales.

1.2.1 Derivada de Orden Dos

Partiendo del desarrollo de Taylor

$$f(x_i + \Delta x) = f(x_i) + \Delta x f'(x_i) + \frac{\Delta x^2}{2!} f''(x_i) + \frac{\Delta x^3}{3!} f'''(x_i) + \frac{\Delta x^4}{4!} f^{(4)}(\xi_p) \quad (1.21)$$

y

$$f(x_i - \Delta x) = f(x_i) - \Delta x f'(x_i) + \frac{\Delta x^2}{2!} f''(x_i) - \frac{\Delta x^3}{3!} f'''(x_i) + \frac{\Delta x^4}{4!} f^{(4)}(\xi_r) \quad (1.22)$$

y eliminado las derivadas primeras, sumando las ecuaciones anteriores y despejando se encuentra que

$$f''(x_i) = \frac{f(x_i - \Delta x) - 2f(x_i) + f(x_i + \Delta x)}{\Delta x^2} - \frac{\Delta x^2}{12} f^{(4)}(\xi_c) \quad (1.23)$$

así, la aproximación a la segunda derivada usando diferencias centradas con un error de truncamiento $O_c(\Delta x^2)$ es

$$f''(x_i) = \frac{f(x_i - \Delta x) - 2f(x_i) + f(x_i + \Delta x)}{\Delta x^2} \quad (1.24)$$

Es común escribir la anterior expresión como

$$f''(x_i) = \frac{f_{i-1} - 2f_i + f_{i+1}}{\Delta x^2}$$

para simplificar notación.

Derivadas Usando Más Puntos Utilizando el valor de la función en más puntos se construyen fórmulas más precisas para las derivadas, algunos ejemplos son

$$\begin{aligned} f''(x_i) &= \frac{f_{i+2} - 2f_{i+1} + f_i}{\Delta x^2} + O(\Delta x) & (1.25) \\ f''(x_i) &= \frac{-f_{i+3} + 4f_{i+2} - 5f_{i+1} + 2f_i}{\Delta x^2} + O(\Delta x^2) \\ f''(x_i) &= \frac{-f_{i+2} + 16f_{i+1} - 30f_i + 16f_{i-1} - f_{i-2}}{12\Delta x^2} + O(\Delta x^4) \end{aligned}$$

1.2.2 Derivadas de Ordenes Mayores

De forma análoga se construyen derivadas de ordenes mayores utilizando el valor de la función en más puntos, algunos ejemplos para derivadas terceras son

$$\begin{aligned} f'''(x_i) &= \frac{f_{i+3} - 3f_{i+2} + 3f_{i+1} - f_i}{\Delta x^3} + O(\Delta x) & (1.26) \\ f'''(x_i) &= \frac{f_{i+2} - 2f_{i+1} + 2f_{i-1} - f_{i-2}}{2\Delta x^3} + O(\Delta x^2) \\ f'''(x_i) &= \frac{f_{i-3} - 8f_{i-2} + 13f_{i-1} - 13f_{i+1} + 8f_{i+2} - f_{i+3}}{8\Delta x^3} + O(\Delta x^4) \end{aligned}$$

Algunos ejemplos para derivadas cuartas son

$$\begin{aligned}
 f''''(x_i) &= \frac{f_{i+4} - 4f_{i+3} + 6f_{i+2} - 4f_{i+1} + f_i}{\Delta x^4} + O(\Delta x) & (1.27) \\
 f''''(x_i) &= \frac{f_{i+2} - 4f_{i+1} + 6f_i - 4f_{i-1} + f_{i-2}}{\Delta x^4} + O(\Delta x^2) \\
 f''''(x_i) &= \frac{-f_{i+3} + 12f_{i+2} - 39f_{i+1} + 56f_i - 39f_{i-1} + 12f_{i-2} - f_{i-3}}{6\Delta x^4} + O(\Delta x^4)
 \end{aligned}$$

1.3 Derivadas en Dos Dimensiones

De forma análoga se construyen aproximaciones en diferencias finitas de primer y segundo orden en dos dimensiones. Usando el teorema de Teylor para funciones en dos variables x y y , es posible escribir de forma exacta para el punto x_i y y_j

$$f(x_i + \Delta x, y_j) = f(x_i, y_j) + \Delta x \frac{\partial f(x_i, y_j)}{\partial x} + \frac{\Delta x}{2} \frac{\partial^2 f(x_i + \theta_1 \Delta x, y_j)}{\partial x^2} \quad (1.28)$$

$$f(x_i, y_j + \Delta y) = f(x_i, y_j) + \Delta y \frac{\partial f(x_i, y_j)}{\partial y} + \frac{\Delta y}{2} \frac{\partial^2 f(x_i, y_j + \theta_2 \Delta y)}{\partial y^2}.$$

Así, la aproximación en diferencias hacia delante de $\partial f/\partial x$ y $\partial f/\partial y$ es

$$\begin{aligned}
 \frac{\partial f(x_i, y_j)}{\partial x} &\simeq \frac{f(x_i + \Delta x, y_j) - f(x_i, y_j)}{\Delta x} & (1.29) \\
 \frac{\partial f(x_i, y_j)}{\partial y} &\simeq \frac{f(x_i, y_j + \Delta y) - f(x_i, y_j)}{\Delta y}
 \end{aligned}$$

o en su forma simplificada (para simplificar la notación, asociamos $\Delta x = h$ y $\Delta y = k$), tenemos

$$\begin{aligned}
 \frac{\partial f(x_i, y_j)}{\partial x} &\simeq \frac{f_{i+1,j} - f_{i,j}}{h} & (1.30) \\
 \frac{\partial f(x_i, y_j)}{\partial y} &\simeq \frac{f_{i,j+1} - f_{i,j}}{k}.
 \end{aligned}$$

La aproximación en diferencias hacia atrás de $\partial f/\partial x$ y $\partial f/\partial y$ es

$$\begin{aligned}
 \frac{\partial f(x_i, y_j)}{\partial x} &\simeq \frac{f(x_i, y_j) - f(x_i - \Delta x, y_j)}{\Delta x} & (1.31) \\
 \frac{\partial f(x_i, y_j)}{\partial y} &\simeq \frac{f(x_i, y_j) - f(x_i, y_j - \Delta y)}{\Delta y}
 \end{aligned}$$

o en su forma simplificada, tenemos

$$\begin{aligned}
 \frac{\partial f(x_i, y_j)}{\partial x} &\simeq \frac{f_{i,j} - f_{i-1,j}}{h} & (1.32) \\
 \frac{\partial f(x_i, y_j)}{\partial y} &\simeq \frac{f_{i,j} - f_{i,j-1}}{k}.
 \end{aligned}$$

La aproximación en diferencias centradas de $\partial f/\partial x$ y $\partial f/\partial y$ es

$$\begin{aligned}\frac{\partial f(x_i, y_j)}{\partial x} &\simeq \frac{f(x_i + \Delta x, y_j) - f(x_i - \Delta x, y_j)}{2\Delta x} \\ \frac{\partial f(x_i, y_j)}{\partial y} &\simeq \frac{f(x_i, y_j + \Delta y) - f(x_i, y_j - \Delta y)}{2\Delta y}\end{aligned}\quad (1.33)$$

o en su forma simplificada, tenemos

$$\begin{aligned}\frac{\partial f(x_i, y_j)}{\partial x} &\simeq \frac{f_{i+1,j} - f_{i-1,j}}{2h} \\ \frac{\partial f(x_i, y_j)}{\partial y} &\simeq \frac{f_{i,j+1} - f_{i,j-1}}{2k}.\end{aligned}\quad (1.34)$$

Por otro lado, la aproximación en diferencias centradas de $\partial^2 f/\partial x^2$ y $\partial^2 f/\partial y^2$ es

$$\begin{aligned}\frac{\partial^2 f(x_i, y_j)}{\partial x^2} &\simeq \frac{f(x_i + \Delta x, y_j) - 2f(x_i, y_j) + f(x_i - \Delta x, y_j)}{\Delta x^2} \\ \frac{\partial^2 f(x_i, y_j)}{\partial y^2} &\simeq \frac{f(x_i, y_j + \Delta y) - f(x_i, y_j) + f(x_i, y_j - \Delta y)}{\Delta y^2}\end{aligned}\quad (1.35)$$

o en su forma simplificada, tenemos

$$\begin{aligned}\frac{\partial^2 f(x_i, y_j)}{\partial x^2} &\simeq \frac{f_{i+1,j} - 2f_{i,j} + f_{i-1,j}}{h^2} \\ \frac{\partial^2 f(x_i, y_j)}{\partial y^2} &\simeq \frac{f_{i,j+1} - 2f_{i,j} + f_{i,j-1}}{k^2}.\end{aligned}\quad (1.36)$$

1.4 Derivadas en Tres Dimensiones

De forma análoga se construyen aproximaciones en diferencias finitas de primer y segundo orden en tres dimensiones. Usando el teorema de Teylor para funciones de tres variables x, y y z , es posible escribir de forma exacta para el punto x_i, y_j y z_k

$$\begin{aligned}f(x_i + \Delta x, y_j, z_k) &= f(x_i, y_j, z_k) + \Delta x \frac{\partial f(x_i, y_j, z_k)}{\partial x} + \frac{\Delta x}{2} \frac{\partial^2 f(x_i + \theta_1 \Delta x, y_j, z_k)}{\partial x^2} \\ f(x_i, y_j + \Delta y, z_k) &= f(x_i, y_j, z_k) + \Delta y \frac{\partial f(x_i, y_j, z_k)}{\partial y} + \frac{\Delta y}{2} \frac{\partial^2 f(x_i, y_j + \theta_2 \Delta y, z_k)}{\partial y^2} \\ f(x_i, y_j, z_k + \Delta z) &= f(x_i, y_j, z_k) + \Delta z \frac{\partial f(x_i, y_j, z_k)}{\partial z} + \frac{\Delta z}{2} \frac{\partial^2 f(x_i, y_j, z_k + \theta_3 \Delta z)}{\partial z^2}\end{aligned}\quad (1.37)$$

Así, la aproximación en diferencias hacia delante de $\partial f/\partial x$, $\partial f/\partial y$ y $\partial f/\partial z$

es

$$\begin{aligned}\frac{\partial f(x_i, y_j, z_k)}{\partial x} &\simeq \frac{f(x_i + \Delta x, y_j, z_k) - f(x_i, y_j, z_k)}{\Delta x} \\ \frac{\partial f(x_i, y_j, z_k)}{\partial y} &\simeq \frac{f(x_i, y_j + \Delta y, z_k) - f(x_i, y_j, z_k)}{\Delta y} \\ \frac{\partial f(x_i, y_j, z_k)}{\partial z} &\simeq \frac{f(x_i, y_j, z_k + \Delta z) - f(x_i, y_j, z_k)}{\Delta z}\end{aligned}\quad (1.38)$$

o en su forma simplificada (para simplificar la notación, asociamos $\Delta x = h$, $\Delta y = l$ y $\Delta z = m$), tenemos

$$\begin{aligned}\frac{\partial f(x_i, y_j, z_k)}{\partial x} &\simeq \frac{f_{i+1,j,k} - f_{i,j,k}}{h} \\ \frac{\partial f(x_i, y_j, z_k)}{\partial y} &\simeq \frac{f_{i,j+1,k} - f_{i,j,k}}{l} \\ \frac{\partial f(x_i, y_j, z_k)}{\partial z} &\simeq \frac{f_{i,j,k+1} - f_{i,j,k}}{m}.\end{aligned}\quad (1.39)$$

La aproximación en diferencias hacia atrás de $\partial f/\partial x$, $\partial f/\partial y$ y $\partial f/\partial z$ es

$$\begin{aligned}\frac{\partial f(x_i, y_j, z_k)}{\partial x} &\simeq \frac{f(x_i, y_j, z_k) - f(x_i - \Delta x, y_j, z_k)}{\Delta x} \\ \frac{\partial f(x_i, y_j, z_k)}{\partial y} &\simeq \frac{f(x_i, y_j, z_k) - f(x_i, y_j - \Delta y, z_k)}{\Delta y} \\ \frac{\partial f(x_i, y_j, z_k)}{\partial z} &\simeq \frac{f(x_i, y_j, z_k) - f(x_i, y_j, z_k - \Delta z)}{\Delta z}\end{aligned}\quad (1.40)$$

o en su forma simplificada, tenemos

$$\begin{aligned}\frac{\partial f(x_i, y_j, z_k)}{\partial x} &\simeq \frac{f_{i,j,k} - f_{i-1,j,k}}{h} \\ \frac{\partial f(x_i, y_j, z_k)}{\partial y} &\simeq \frac{f_{i,j,k} - f_{i,j-1,k}}{l} \\ \frac{\partial f(x_i, y_j, z_k)}{\partial z} &\simeq \frac{f_{i,j,k} - f_{i,j,k-1}}{m}.\end{aligned}\quad (1.41)$$

La aproximación en diferencias centradas de $\partial f/\partial x$, $\partial f/\partial y$ y $\partial f/\partial z$ es

$$\begin{aligned}\frac{\partial f(x_i, y_j, z_k)}{\partial x} &\simeq \frac{f(x_i + \Delta x, y_j, z_k) - f(x_i - \Delta x, y_j, z_k)}{2\Delta x} \\ \frac{\partial f(x_i, y_j, z_k)}{\partial y} &\simeq \frac{f(x_i, y_j + \Delta y, z_k) - f(x_i, y_j - \Delta y, z_k)}{2\Delta y} \\ \frac{\partial f(x_i, y_j, z_k)}{\partial z} &\simeq \frac{f(x_i, y_j, z_k + \Delta z) - f(x_i, y_j, z_k - \Delta z)}{2\Delta z}\end{aligned}\quad (1.42)$$

o en su forma simplificada, tenemos

$$\begin{aligned}\frac{\partial f(x_i, y_j, z_k)}{\partial x} &\simeq \frac{f_{i+1,j,k} - f_{i-1,j,k}}{2h} \\ \frac{\partial f(x_i, y_j, z_k)}{\partial y} &\simeq \frac{f_{i,j+1,k} - f_{i,j-1,k}}{2l} \\ \frac{\partial f(x_i, y_j, z_k)}{\partial z} &\simeq \frac{f_{i,j,k+1} - f_{i,j,k-1}}{2m}.\end{aligned}\tag{1.43}$$

Por otro lado, la aproximación en diferencias centradas de $\partial^2 f / \partial x^2$, $\partial^2 f / \partial y^2$ y $\partial^2 f / \partial z^2$ es

$$\begin{aligned}\frac{\partial^2 f(x_i, y_j, z_k)}{\partial x^2} &\simeq \frac{f(x_i + \Delta x, y_j, z_k) - 2f(x_i, y_j, z_k) + f(x_i - \Delta x, y_j, z_k)}{\Delta x^2} \\ \frac{\partial^2 f(x_i, y_j, z_k)}{\partial y^2} &\simeq \frac{f(x_i, y_j + \Delta y, z_k) - f(x_i, y_j, z_k) + f(x_i, y_j - \Delta y, z_k)}{\Delta y^2} \\ \frac{\partial^2 f(x_i, y_j, z_k)}{\partial z^2} &\simeq \frac{f(x_i, y_j, z_k + \Delta z) - f(x_i, y_j, z_k) + f(x_i, y_j, z_k - \Delta z)}{\Delta z^2}\end{aligned}\tag{1.44}$$

o en su forma simplificada, tenemos

$$\begin{aligned}\frac{\partial^2 f(x_i, y_j, z_k)}{\partial x^2} &\simeq \frac{f_{i+1,j,k} - 2f_{i,j,k} + f_{i-1,j,k}}{h^2} \\ \frac{\partial^2 f(x_i, y_j, z_k)}{\partial y^2} &\simeq \frac{f_{i,j+1,k} - 2f_{i,j,k} + f_{i,j-1,k}}{l^2} \\ \frac{\partial^2 f(x_i, y_j, z_k)}{\partial z^2} &\simeq \frac{f_{i,j,k+1} - 2f_{i,j,k} + f_{i,j,k-1}}{m^2}.\end{aligned}\tag{1.45}$$

2 Método de Diferencias Finitas en Una Dimensión

Consideremos la ecuación diferencial parcial

$$(p(x)u'(x))' + q(x)u'(x) - r(x)u(x) = f(x) \quad (2.1)$$

$$\text{en } a \leq x \leq b \quad \text{donde: } u(a) = u_\alpha \text{ y } u(b) = u_\beta$$

con condiciones de frontera Dirichlet o cualquier otro tipo de condiciones de frontera. Para usar el procedimiento general de solución numérica mediante el método de diferencias finitas, debemos de hacer:

1. Generar una malla del dominio, i.e. una malla es un conjunto finito de puntos en los cuales buscaremos la solución aproximada a la ecuación diferencial parcial.
2. Sustituir las derivadas correspondiente con alguna de las formulas de diferencias finitas centradas (véase secciones 1.1 y 1.2), en cada punto donde la solución es desconocida para obtener un sistema algebraico de ecuaciones $\underline{A}u = \underline{f}$.
3. Resolver el sistema de ecuaciones (véase capítulo 3), y así obtener la solución aproximada en cada punto de la malla.

2.1 Problema con Condiciones de Frontera Dirichlet

Consideremos un caso particular de la Ec.(2.1) definido por la ecuación

$$u''(x) = f(x), \quad 0 \leq x \leq 1, \quad u(0) = u_\alpha, \quad u(1) = u_\beta \quad (2.2)$$

con condiciones de frontera Dirichlet. Para usar el procedimiento general de solución numérica mediante el método de diferencias finitas, debemos de hacer:

1. Generamos una malla homogénea del dominio³

$$x_i = ih, \quad i = 0, 1, \dots, n, \quad h = \frac{1}{n} = \Delta x \quad (2.3)$$

2. Sustituimos la derivada con Ec.(1.24) en cada punto donde la solución es desconocida para obtener un sistema algebraico de ecuaciones. Así, en cada punto x_i de la malla aproximamos la ecuación diferencial por⁴

$$u''(x_i) \approx \frac{u(x_i - h) - 2u(x_i) + u(x_i + h)}{h^2} \quad (2.4)$$

³En el caso de que la malla no sea homogénea, es necesario incluir en la derivada a que h se refiere, por ejemplo en cada punto i , tenemos la h_{i-} (por la izquierda) y la h_{i+} (por la derecha), i.e. $u''(x_i) \approx \frac{u(x_i - h_{i-}) - 2u(x_i) + u(x_i + h_{i+})}{(h_{i-})(h_{i+})}$.

⁴Notemos que en cada punto de la malla, la aproximación por diferencias finitas supone la solución de tres puntos de la malla x_{i-1}, x_i y x_{i+1} . Al conjunto de estos tres puntos de la malla son comúnmente llamados el estencil de diferencias finitas.

o en su forma simplificada

$$u''(x_i) \approx \frac{u_{i-1} - 2u_i + u_{i+1}}{h^2} \quad (2.5)$$

definiendo la solución aproximada de $u(x)$ en x_i como u_i como la solución del siguiente sistema lineal de ecuaciones

$$\begin{aligned} \frac{u_\alpha - 2u_1 + u_2}{h^2} &= f(x_1) \\ \frac{u_1 - 2u_2 + u_3}{h^2} &= f(x_2) \\ &\vdots \\ \frac{u_{i-1} - 2u_i + u_{i+1}}{h^2} &= f(x_i) \\ &\vdots \\ \frac{u_{n-3} - 2u_{n-2} + u_{n-1}}{h^2} &= f(x_{n-2}) \\ \frac{u_{n-2} - 2u_{n-1} + u_\beta}{h^2} &= f(x_{n-1}). \end{aligned} \quad (2.6)$$

Este sistema de ecuaciones se puede escribir como un matriz \underline{A} y los vectores \underline{u} y \underline{f} de la forma

$$\begin{bmatrix} -\frac{2}{h^2} & \frac{1}{h^2} & & & & & & \\ \frac{1}{h^2} & -\frac{2}{h^2} & \frac{1}{h^2} & & & & & \\ & \frac{1}{h^2} & -\frac{2}{h^2} & \frac{1}{h^2} & & & & \\ & & \ddots & \ddots & \ddots & & & \\ & & & \frac{1}{h^2} & -\frac{2}{h^2} & \frac{1}{h^2} & & \\ & & & & \frac{1}{h^2} & -\frac{2}{h^2} & & \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_{n-2} \\ u_{n-1} \end{bmatrix} = \begin{bmatrix} f(x_1) - \frac{u_\alpha}{h^2} \\ f(x_2) \\ f(x_3) \\ \vdots \\ f(x_{n-2}) \\ f(x_{n-1}) - \frac{u_\beta}{h^2} \end{bmatrix}.$$

factorizando $1/h^2$ del sistema lineal $\underline{A}u = \underline{f}$, tenemos

$$\frac{1}{h^2} \begin{bmatrix} -2 & 1 & & & & & & \\ 1 & -2 & 1 & & & & & \\ & 1 & -2 & 1 & & & & \\ & & \ddots & \ddots & \ddots & & & \\ & & & 1 & -2 & 1 & & \\ & & & & 1 & -2 & & \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_{n-2} \\ u_{n-1} \end{bmatrix} = \begin{bmatrix} f(x_1) - \frac{u_\alpha}{h^2} \\ f(x_2) \\ f(x_3) \\ \vdots \\ f(x_{n-2}) \\ f(x_{n-1}) - \frac{u_\beta}{h^2} \end{bmatrix}.$$

esta última forma de expresar el sistema lineal algebraico asociado es preferible para evitar problemas numéricos al momento de resolver el sistema lineal por métodos iterativos (véase capítulo 3.2) principalmente cuando $h \rightarrow 0$.

3. Resolviendo el sistema de ecuaciones (véase capítulo 3), obtenemos la solución aproximada en cada punto interior de la malla. La solución completa al problema la obtenemos al formar el vector

$$\left[u_\alpha \quad u_1 \quad u_2 \quad u_3 \quad \cdots \quad u_{n-2} \quad u_{n-1} \quad u_\beta \right].$$

Para el problema general dado por la Ec.(2.1)

$$(p(x) u'(x))' + q(x) u'(x) - r(x) u(x) = f(x) \quad (2.7)$$

$$\text{en } a \leq x \leq b \quad \text{donde: } u(a) = u_\alpha \text{ y } u(b) = u_\beta$$

el término $q(x) u'(x)$ algunas veces es llamado el término advectivo⁵ si u es la velocidad. Existen dos diferentes técnicas de discretización que se detallan a continuación:

- Si las funciones $p(x)$, $q(x)$ y $r(x)$ son constantes, el esquema de diferencias finitas centradas para todas las derivadas, este está dado por el estencil

$$p_i \frac{u_{i-1} - 2u_i + u_{i+1}}{h^2} + q_i \frac{u_{i+1} - u_{i-1}}{2h} - r_i u_i = f_i \quad i = 1, 2, \dots, n. \quad (2.8)$$

donde la ventaja de esta discretización, es que el método es de segundo orden de exactitud. La desventaja es que los coeficientes de la matriz generada pueden no ser diagonal dominante si $r(x) > 0$ y $p(x) > 0$. Cuando la advección $|p(x)|$ es grande, la ecuación se comporta como la ecuación de onda.

- Tomando las funciones $p(x, t)$, $q(x, t)$ y $r(x, t)$ más generales posibles, es necesario hacer una discretización que garantice que el método es de segundo orden de exactitud, esto se logra mediante la siguiente discretización para $(p(x, t) u'(x, t))'$ mediante

$$\begin{aligned} \frac{\partial}{\partial x} \left(p \frac{\partial u}{\partial x} \right) (x, t) &\simeq \left[p \left(x + \frac{\Delta x}{2}, t \right) \frac{u(x + \Delta x, t) - u(x, t)}{\Delta x} \right. \\ &\quad \left. - p \left(x - \frac{\Delta x}{2}, t \right) \frac{u(x, t) - u(x - \Delta x, t)}{\Delta x} \right] / \Delta x \end{aligned} \quad (2.9)$$

entonces se tiene que

$$\frac{p(u_{i+1}, t) \frac{u_{i+1} + u_i}{h} - p(u_{i-1}, t) \frac{u_i - u_{i-1}}{h}}{h} + q_i \frac{u_{i+1} - u_{i-1}}{2h} - r_i u_i = f_i \quad (2.10)$$

para $i = 1, 2, \dots, n$, (véase [51] pág. 78 y 79).

⁵ Cuando la advección es fuerte, esto es cuando $|q(x)|$ es grande, la ecuación se comporta como si fuera una ecuación de onda.

- El esquema mixto, en donde se usa el esquema de diferencias finitas centradas para el término de difusión y el esquema *upwind* para el término de advección

$$\begin{aligned}
 p_i \frac{u_{i-1} - 2u_i + u_{i+1}}{h^2} + q_i \frac{u_{i+1} - u_{i-1}}{h} - r_i u_i &= f_i, \text{ si } q_i \geq 0 \\
 p_i \frac{u_{i-1} - 2u_i + u_{i+1}}{h^2} + q_i \frac{u_i - u_{i-1}}{h} - r_i u_i &= f_i, \text{ si } q_i < 0
 \end{aligned}
 \tag{2.11}$$

el propósito es incrementar el dominio de la diagonal. Este esquema es de orden uno de exactitud y es altamente recomendable su uso si $|q(x)| \sim 1/h$, en caso de no usarse, se observará que la solución numérica oscila alrededor del cero.

Veamos unos ejemplos desarrollados en SCILAB⁶

Ejemplo 1 *Sea*

$$-u''(x) + u(x) = 0, \quad 0 \leq x \leq 1, \quad u(0) = 0, \quad u(1) = 1$$

entonces el programa queda implementado como:

```

a=0; // Inicio dominio
c=1; // Fin dominio
M=50; // Partición
N=M-2; // Nodos interiores
h=(c-a)/(M-1); // Incremento en la malla
Y0=0; // Condición inicial en el inicio del dominio
Y1=1; // Condición inicial en el fin del dominio
A=zeros(N,N); // Matriz A
b=zeros(N); // Vector b

P=2/(h^2);
Q=-1/(h^2)+1/(2*h);
R=-1/(h^2)-1/(2*h);

// Primer renglon de la matriz A y vector b
A(1,1)=P;
A(1,2)=Q;
b(1)=-Y0*R;
// Renglones intermedios de la matriz A y vector b
for i=2:N-1

```

⁶Scilab es un programa open source para el cálculo numérico el cual provee un poderoso ambiente de cálculo para aplicaciones Científicas y de Ingeniería [<http://www.scilab.org>].

```
A(i,i-1)=R;
A(i,i)=P;
A(i,i+1)=Q;
end
// Relglon final de la matriz A y vector b
A(N,N-1)=R;
A(N,N)=P;
b(N)=-Y1*Q;

// Resuleve el sistema lineal Ax=b
x=inv(A)*b;

// Prepara la graficación
xx=zeros(M,1);
for i=1:M
    xx(i)=a+h*(i-1);
end
yy=zeros(M,1);
yy(1)=Y0; // Condición inicial
for i=1:N
    yy(i+1)=x(i);
end
yy(M)=Y1; // Condición inicial
// Grafica la solución de la Ecuación Diferencial Parcial en 1D
plot2d(xx,yy)
```

Ejemplo 2 Sea

$$u''(x) = \pi^2 \cos(\pi x), \quad 0 \leq x \leq 1, \quad u(0) = 1, \quad u(1) = -1$$

entonces el programa queda implementado como:

```
function y=LadoDerecho(x)
    y=-%pi*%pi*cos(%pi*x);
endfunction

function y=SolucionAnalitica(x)
    y=cos(%pi*x);
endfunction

a=-1; // Inicio dominio
c=2; // Fin dominio
M=100; // Partición
N=M-2; // Nodos interiores
h=(c-a)/(M-1); // Incremento en la malla
Y0=-1; // Condición inicial en el inicio del dominio
```

```
Y1=1; // Condición inicial en el fin del dominio

A=zeros(N,N); // Matriz A
b=zeros(N); // Vector b
R=1/(h^2);
P=-2/(h^2);
Q=1/(h^2);
// Primer renglon de la matriz A y vector b
A(1,1)=P;
A(1,2)=Q;
b(1)=LadoDerecho(a)-Y0*R;
// Renglones intermedios de la matriz A y vector b
for i=2:N-1
    A(i,i-1)=R;
    A(i,i)=P;
    A(i,i+1)=Q;
    b(i)=LadoDerecho(a+h*(i-1));
end
// Renglón final de la matriz A y vector b
A(N,N-1)=R;
A(N,N)=P;
b(N)=LadoDerecho(a+h*N)-Y1*Q;

// Resuelve el sistema lineal Ax=b
x=inv(A)*b;

// Prepara la graficación
xx=zeros(M,1);
zz=zeros(M,1);
for i=1:M
    xx(i)=a+h*(i-1);
    zz(i)=SolucionAnalitica(xx(i));
end
yy=zeros(M,1);
yy(1)=Y0; // Condición inicial
for i=1:N
    yy(i+1)=x(i);
end
yy(M)=Y1; // Condición inicial
// Grafica la solución de la Ecuación Diferencial Parcial en 1D
plot2d(xx,[yy,zz])
```


2.2 Problema con Condiciones de Frontera Neumann

Consideremos el problema

$$u''(x) = f(x), \quad 0 \leq x \leq 1 \quad (2.12)$$

con condiciones de frontera Neumann

$$\frac{du}{dx} = cte_1 \text{ en } u(0) \text{ y } \frac{du}{dx} = cte_2 \text{ en } u(1)$$

para usar el procedimiento general de solución numérica mediante el método de diferencias finitas, primeramente debemos de discretizar las condiciones de frontera, una manera seria usar para la primera condición de frontera una aproximación usando diferencias progresivas Ec.(1.5)

$$\left. \frac{du}{dx} \right|_{x_i} = \frac{u(x_i + h) - u(x_i)}{h}$$

quedando

$$\frac{u_1 - u_0}{h} = cte_1 \quad (2.13)$$

para la segunda condición de frontera una aproximación usando diferencias regresivas Ec.(1.10)

$$\left. \frac{du}{dx} \right|_{x_i} = \frac{u(x_i) - u(x_i - h)}{h}$$

quedando

$$\frac{u_n - u_{n-1}}{h} = cte_2 \quad (2.14)$$

pero el orden de aproximación no seria el adecuado pues estamos aproximando el dominio con diferencias centradas con un error local de truncamiento de segundo orden $O_c(\Delta x^2)$, en lugar de ello usaremos diferencias centradas Ec.(1.15) para tener todo el dominio con el mismo error local de truncamiento.

Para usar diferencias centradas Ec.(1.15)

$$\left. \frac{du}{dx} \right|_{x_i} = \frac{u(x_i + h) - u(x_i - h)}{2h}$$

en el primer nodo necesitamos introducir un punto de la malla ficticio $x_{-1} = (x_0 - \Delta x)$ con un valor asociado a u_{-1} , entonces

$$\frac{u_1 - u_{-1}}{2h} = cte_1 \quad (2.15)$$

así también, en el último nodo necesitamos introducir un punto de la malla ficticio $x_{n+1} = (x_n + \Delta x)$ con un valor asociado a u_{n+1} , obteniendo

$$\frac{u_{n+1} - u_{n-1}}{2h} = cte_2. \quad (2.16)$$

Estos valores no tienen significado físico alguno, dado que esos puntos se encuentran fuera del dominio del problema. Entonces debemos de hacer:

1. Generamos una malla homogénea del dominio

$$x_i = ih, \quad i = 0, 1, \dots, n, \quad h = \frac{1}{n} = \Delta x. \quad (2.17)$$

2. Sustituimos la derivada con Ec.(1.24) en cada punto donde la solución es desconocida para obtener un sistema algebraico de ecuaciones. Así, en cada punto x_i de la malla aproximamos la ecuación diferencial por

$$u''(x_i) \approx \frac{u(x_i - h) - 2u(x_i) + u(x_i + h)}{h^2} \quad (2.18)$$

definiendo la solución aproximada de $u(x)$ en x_i como u_i como la solución del siguiente sistema lineal de ecuaciones

$$\begin{aligned} \frac{u_1 - u_{-1}}{2h} &= cte_1 \\ \frac{u_0 - 2u_1 + u_2}{h^2} &= f(x_1) \\ &\vdots \\ \frac{u_{i-1} - 2u_i + u_{i+1}}{h^2} &= f(x_i) \\ &\vdots \\ \frac{u_{n-2} - 2u_{n-1} + u_n}{h^2} &= f(x_{n-1}) \\ \frac{u_{n+1} - u_{n-1}}{2h} &= cte_2. \end{aligned} \quad (2.19)$$

3. Resolviendo el sistema de ecuaciones (véase capítulo 3), obtenemos la solución aproximada en cada punto de la malla.

2.3 Problema con Condiciones de Frontera Robin

El método de un punto de la malla ficticio es usado para el manejo de las condiciones de frontera mixtas, también conocidas como condiciones de frontera Robin. Sin pérdida de generalidad, supongamos que en $x = a$, tenemos

$$\alpha u'(a) + \beta u(b) = \gamma$$

donde $\alpha \neq 0$. Entonces usando el punto de la malla ficticio, tenemos que

$$\alpha \frac{u_1 - u_{-1}}{2h} + \beta u_n = \gamma$$

o

$$u_{-1} = u_1 + \frac{2\beta}{\alpha} u_n - \frac{2h\gamma}{\alpha}$$

introduciendo esto en términos de diferencias finitas centradas, en $x = x_0$, entonces se tiene que

$$\left(-\frac{2}{h^2} + \frac{2\beta}{\alpha h}\right) u_n + \frac{2}{h^2} u_1 = f_0 + \frac{2\gamma}{\alpha h}$$

o

$$\left(-\frac{1}{h^2} + \frac{\beta}{\alpha h}\right) u_n + \frac{1}{h^2} u_1 = \frac{f_0}{2} + \frac{\gamma}{\alpha h}$$

lo que genera coeficientes simétricos en la matriz.

Consideremos el problema

$$u''(x) = f(x), \quad 0 \leq x \leq 1 \quad (2.20)$$

con condiciones de frontera Dirichlet y Neumann

$$u(0) = u_\alpha \quad \text{y} \quad \frac{du}{dx} = cte_1 \text{ en } u(1).$$

respectivamente. Para usar el procedimiento general de solución numérica mediante el método de diferencias finitas, primeramente debemos de expresar la condición de frontera Neumann mediante diferencias centradas Ec.(1.15)

$$\left. \frac{du}{dx} \right|_{x_i} = \frac{u(x_i + h) - u(x_i - h)}{2h}$$

en el último nodo necesitamos introducir un punto de la malla ficticio $x_{n+1} = (x_n + \Delta x)$ con un valor asociado a u_{n+1} quedando

$$\frac{u_{n+1} - u_{n-1}}{2h} = cte_2. \quad (2.21)$$

Este valor no tiene significado físico alguno, dado que este punto se encuentra fuera del dominio del problema.

Entonces debemos de hacer:

1. Generamos una malla homogénea del dominio

$$x_i = ih, \quad i = 0, 1, \dots, n, \quad h = \frac{1}{n} = \Delta x. \quad (2.22)$$

2. Sustituimos la derivada con Ec.(1.24) en cada punto donde la solución es desconocida para obtener un sistema algebraico de ecuaciones. Así, en cada punto x_i de la malla aproximamos la ecuación diferencial por

$$u''(x_i) \approx \frac{u(x_i - h) - 2u(x_i) + u(x_i + h)}{h^2} \quad (2.23)$$

definiendo la solución aproximada de $u(x)$ en x_i como u_i como la solución del siguiente sistema lineal de ecuaciones

$$\begin{aligned}
 \frac{u_\alpha - 2u_1 + u_2}{h^2} &= f(x_1) \\
 \frac{u_1 - 2u_2 + u_3}{h^2} &= f(x_2) \\
 &\vdots \\
 \frac{u_{i-1} - 2u_i + u_{i+1}}{h^2} &= f(x_i) \\
 &\vdots \\
 \frac{u_{n-2} - 2u_{n-1} + u_n}{h^2} &= f(x_{n-1}) \\
 \frac{u_{n+1} - u_{n-1}}{2h} &= cte_1.
 \end{aligned} \tag{2.24}$$

3. Resolviendo el sistema de ecuaciones (véase capítulo 3), obtenemos la solución aproximada en cada punto de la malla. La solución completa al problema la obtenemos al formar el vector

$$[u_\alpha \quad u_1 \quad u_2 \quad u_3 \quad \cdots \quad u_{n-2} \quad u_{n-1} \quad u_n].$$

Veamos unos ejemplos desarrollados en SCILAB⁷

Ejemplo 3 Sea

$$u''(x) = -\pi^2 \cos(\pi x), \quad 0 \leq x \leq 0.5, \quad u(0) = 1, \quad u'(0.5) = -\pi$$

entonces el programa queda implementado como:

```
function y=LadoDerecho(x)
y=-%pi*%pi*cos(%pi*x);
endfunction
```

```
function y=SolucionAnalitica(x)
y=cos(%pi*x);
endfunction
```

```
a=0; // Inicio dominio
c=0.5; // Fin dominio
M=40; // Partición
N=M-1; // Nodos interiores
h=(c-a)/(M-1); // Incremento en la malla
Y0=1; // Condición Dirchlet inicial en el inicio del dominio
```

⁷Scilab es un programa open source para el cálculo numérico el cual provee un poderoso ambiente de cálculo para aplicaciones Científicas y de Ingeniería [<http://www.scilab.org>].

```
Y1=-%pi; // Condición Neumann inicial en el fin del dominio
A=zeros(N,N); // Matriz A
b=zeros(N); // Vector b

R=1/(h^2);
P=-2/(h^2);
Q=1/(h^2);

// Primer renglon de la matriz A y vector b
A(1,1)=P;
A(1,2)=Q;
b(1)=LadoDerecho(a)-Y0*R; // Frontera Dirichlet
// Renglones intermedios de la matriz A y vector b
for i=2:N-1
    A(i,i-1)=R;
    A(i,i)=P;
    A(i,i+1)=Q;
    b(i)=LadoDerecho(a+h*(i-1));
end
// Renglón final de la matriz A y vector b
A(N,N-1)=-1/(h^2);
A(N,N)=-1/(h^2);
b(N)=Y1/h; // Frontera Neumann

// Resuelve el sistema lineal Ax=b
x=inv(A)*b;

// Prepara la graficación
xx=zeros(M,1);
zz=zeros(M,1);
for i=1:M
    xx(i)=a+h*(i-1);
    zz(i)=SolucionAnalitica(xx(i));
end
yy=zeros(M,1);
yy(1)=Y0; // Condición inicial
for i=1:N
    yy(i+1)=x(i);
end
// Grafica la solución de la Ecuación Diferencial Parcial en 1D
plot2d(xx,[yy,zz])
```

Ejemplo 4 *Sea*

$$-u''(x) - k^2u(x) = 0, \quad 0 \leq x \leq 1, \quad u(0) = 1, \quad u'(1) = iku(1)$$

con $k = 150$, entonces el programa queda implementado como⁸:

*TEST = 1; // (0) Diferencias finitas, (1) Diferencias finitas exactas segun
Yau Shu Wong y Guangrui Li*

```
function y=LadoDerecho(x)
y=0.0;
endfunction
```

```
function y=SolucionAnalitica(x, k)
//y=cos(k*x)+%i*sin(k*x);
y=exp(%i*k*x);
endfunction
```

```
K = 150;
KK = K*K;
a=0; // Inicio dominio
c=1; // Fin dominio
M=300; // Partición
N=M-1; // Nodos interiores
h=(c-a)/(M-1); // Incremento en la malla
Y0=1; // Condición Dirchlet inicial en el inicio del dominio
Y1=%i*K; // Condición Neumann inicial en el fin del dominio
```

```
A=zeros(N,N); // Matriz A
b=zeros(N); // Vector b
```

```
if TEST = 0 then
R=-1/(h^2);
P=2/(h^2)-KK;
Q=-1/(h^2);
else
R=-1/(h^2);
P=(2*cos(K*h)+(K*h)^2)/(h^2) - KK;
Q=-1/(h^2);
end
```

8

Esta ecuación se conoce como la ecuación de Helmholtz la cual es difícil de resolver si $r(x) \leq 0$ y $|r(x)|$ es grande, i.e. $r(x) \sim 1/h^2$. Para resolver dicha ecuación, se usa diferencias finitas y diferencias finitas exactas, procedimiento desarrollado en: Exact Finite Difference Schemes for Solving Helmholtz equation at any wavenumber, Yau Shu Wong and Guangrui Lim International Journal of Numerical Analysis and Modeling, Volumen 2, Number 1, Pages 91-108, 2011.

```
// Primer renglon de la matriz A y vector b
A(1,1)=P;
A(1,2)=Q;
b(1)=LadoDerecho(a)-Y0*R; // Frontera dirichlet
// Renglones intermedios de la matriz A y vector b
for i=2:N-1
    A(i,i-1)=R;
    A(i,i)=P;
    A(i,i+1)=Q;
    b(i)=LadoDerecho(a+h*(i-1));
end
// Renglón final de la matriz A y vector b
if TEST = 0 then
    A(N,N-1)=1/(h^2);
    A(N,N)=-1/(h^2)+ Y1/h;
    b(N)=LadoDerecho(c)/2;
else
    A(N,N-1)=1/(h^2);
    A(N,N)=-1/(h^2)+ %i*sin(K*h)/(h^2);
    b(N)=LadoDerecho(c)/2;
end

// Resuelve el sistema lineal Ax=b
x=inv(A)*b;

ESC = 5;
xxx=zeros(M*ESC,1);
zzz=zeros(M*ESC,1);
for i=1:M*ESC
    xxx(i)=a+h/ESC*(i-1);
    zzz(i)=SolucionAnalitica(xxx(i),K);
end
// Prepara la graficación
xx=zeros(M,1);
zz=zeros(M,1);
for i=1:M
    xx(i)=a+h*(i-1);
    zz(i)=SolucionAnalitica(xx(i),K);
end
yy=zeros(M,1);
yy(1)=Y0; // Condición inicial
for i=1:N
    yy(i+1)=x(i);
end
// Gráfica la solución de la Ecuación Diferencial Parcial en 1D
plot2d(xx,yy,15)
```

plot2d(xxx,zzz)

2.4 Discretización del Tiempo

Hasta ahora se ha visto como discretizar la parte espacial de las ecuaciones diferenciales parciales, lo cual nos permite encontrar la solución estática de los problemas del tipo elíptico. Sin embargo, para ecuaciones del tipo parabólico e hiperbólico dependen del tiempo, se necesita introducir una discretización a las derivadas con respecto del tiempo. Al igual que con las discretizaciones espaciales, podemos utilizar algún esquema de diferencias finitas en la discretización del tiempo.

2.4.1 Ecuación con Primera Derivada Temporal

Para la solución de la ecuaciones diferenciales con derivada temporal (u_t), se emplean diferentes esquemas en diferencias finitas para la discretización del tiempo. Estos esquemas se conocen de manera general como esquemas $theta(\theta)$.

Definiendo la ecuación diferencial parcial general de segundo orden como

$$u_t = \mathcal{L}u \quad (2.25)$$

donde

$$\mathcal{L}u = (p(x)u'(x))' + q(x)u'(x) - r(x)u(x) - f(x) \quad (2.26)$$

aquí, los coeficientes p, q y r pueden depender del espacio y del tiempo. Entonces el esquema $theta$ está dado por

$$u_t = (1 - \theta)(\mathcal{L}u)^j + \theta(\mathcal{L}u)^{j+1}. \quad (2.27)$$

Existen diferentes casos del esquema $theta$ a saber:

- Para $\theta = 0$, se obtiene un esquema de diferencias finitas hacia delante en el tiempo, conocido como esquema completamente explícito, ya que el paso $n + 1$ se obtiene de los términos del paso anterior n . Es un esquema sencillo, el cual es condicionalmente estable cuando $\Delta t \leq \frac{\Delta x^2}{2}$.
- Para $\theta = 1$, se obtiene el esquema de diferencias finitas hacia atrás en el tiempo, conocido como esquema completamente implícito, el cual es incondicionalmente estable.
- Para $\theta = \frac{1}{2}$, se obtiene un esquema de diferencias finitas centradas en el tiempo, conocido como esquema Crank-Nicolson, este esquema es también incondicionalmente estable y es más usado por tal razón.

Para la implementación del esquema Crank-Nicolson se toma una diferencia progresiva para el tiempo y se promedian las diferencias progresivas y regresivas

en el tiempo para las derivadas espaciales. Entonces si tenemos la Ec. (2.26), las discretizaciones correspondientes son:

$$u_t \simeq \frac{u_i^{j+1} - u_i^j}{\Delta t} \quad (2.28)$$

$$(p(x) u'(x))' \simeq \frac{p}{2} \left[\frac{u_{i-1}^j - 2u_i^j + u_{i+1}^j}{\Delta x^2} + \frac{u_{i-1}^{j+1} - 2u_i^{j+1} + u_{i+1}^{j+1}}{\Delta x^2} \right] \quad (2.29)$$

$$q(x) u'(x) \simeq \frac{q}{2} \left[\frac{u_{i-1}^j + u_{i+1}^j}{2\Delta x} + \frac{u_{i-1}^{j+1} + u_{i+1}^{j+1}}{2\Delta x} \right] \quad (2.30)$$

además de $r(x)u_i^j$ y f_i^j .

Entonces, una vez que se sustituyen las derivadas por su forma en diferencias finitas, lo siguiente es formar el sistema

$$Au^{j+1} = Bu^j + f^j \quad (2.31)$$

esto se logra, colocando del lado izquierdo la igualdad de los términos que contengan el paso del tiempo correspondiente a $j + 1$ y del lado derecho a los correspondientes términos de j .

A continuación, veamos un ejemplo del esquema Crank-Nicolson desarrollados en SCILAB⁹

Ejemplo 5 *Sea*

$$u_t - a(x)u''(x) - b(x)u'(x) + c(x)u = f, \quad l_0 \leq x \leq l, \quad 0 < t < T$$

entonces el programa queda implementado como:

```
// Crank-Nicolson
// Para una EDP de segundo orden
// u_t + a(x)u_xx + b(x)u_x + c(x)u = f
// Dominio
// l0 < x < l
// 0 < t < T
// Condiciones de frontera Dirichlet
// u(0,t) = u(l,t) = constante 0 < t < T cond de frontera
// Condicion inicial
// u(x,0) = g(x) l0 <= x <= l
// Datos de entrada
// intervalo [l0, l]
// entero m >= 3
// entero N >= 1
// Salida
// aproximaciones w_ij a u(x_i, t_j)
```

⁹Scilab es un programa open source para el cálculo numérico el cual provee un poderoso ambiente de cálculo para aplicaciones Científicas y de Ingeniería [<http://www.scilab.org>].

```
// Funciones de los coeficientes
function y = a(x)
y = -1;
endfunction
function y = b(x)
y = 0;
endfunction
function y = c(x)
y = 0;
endfunction
function y = f(x)
y = 0;
endfunction
// funcion de la condicion inicial
function y = condicion_inicial(x)
y = sin(x * %pi);
endfunction
// Condiciones de frontera
// Solo Dirichlet
cond_izq = 0;
cond_der = 0;
// Datos de entrada
l0 = 0;l = 1; // intervalo [0,1]
m = 11; // Numero de nodos
M = m - 2; // Nodos interiores
// Division del espacio y del tiempo
h = (l - l0)/(m-1);
k = 0.025; // Paso del tiempo
N = 50; // Numero de iteraciones
//  $Aw^{(j+1)} = Bw^{(j)} + f^{(j)}$ 
// creo el vector w donde se guardara la solucion para cada tiempo
// A matriz del lado izquierdo
// B matriz del lado derecho
// B_prima para guardar el resultado de  $Bw^{(j)}$ 
// ff que es el vector de los valores de f en cada nodo
w = zeros(M,1);
ff = zeros(M,1);
A = zeros(M,M);
B = zeros(M,M);
//B_prima = zeros(M,1);
w_sol = zeros(m,m)
// Se crea el espacio de la solucion o malla
espacio = zeros(M,1)
for i = 1:m
xx = l0 + (i-1)*h;
espacio(i) = xx;
```

```

end
disp(espacio, "Espacio ")
// Condicion inicial
for i=1:M
w(i) = condicion_inicial(espacio(i+1));
end
w_sol(1) = cond_izq;
for kk = 1:M
w_sol(kk + 1) = w(kk);
end
w_sol(m) = cond_izq;
plot(espacio, w_sol);
disp(w, "Condiciones iniciales")
// primer renglon de cada matriz
A(1,1) = 1/k - a(l0 + h)/(h*h);
A(1,2) = a(l0 + h)/(2*h*h) + b(l0 + h)/(4*h) ;
B(1,1) = 1/k + a(l0 + h)/(h*h) - c(l0 + h);
B(1,2) = - a(l0 + h)/(2*h*h) - b(l0 + h)/(4*h);
ff(1) = f(l0 + h) - cond_izq;
// se completa las matrices desde el renglon 2 hasta el m-2
for i = 2:M-1
A(i, i-1) = a(l0 + i*h)/(2*h*h) - b(l0 + i*h)/(4*h) ;
A(i,i) = 1/k - a(l0 + i*h)/(h*h);
A(i,i+1) = a(l0 + i*h)/(2*h*h) + b(l0 + i*h)/(4*h) ;
B(i, i-1) = - a(l0 + i*h)/(2*h*h) + b(l0 + i*h)/(4*h);
B(i,i) = 1/k + a(l0 + i*h)/(h*h) - c(l0 + i*h);
B(i,i+1) = - a(l0 + i*h)/(2*h*h) - b(l0 + i*h)/(4*h);
end
// Ultimo renglon de cada matriz
A(M,M-1) = a(l - h)/(2*h*h) - b(l-h)/(4*h) ;
A(M,M) = 1/k - a(l - h)/(h*h);
B(M,M-1) = - a(l-h)/(2*h*h) + b(l-h)/(4*h);
B(M,M) = 1/k + a(l-h)/(h*h) - c(l-h);
ff(M) = f(l - h) - cond_der;
// Resolvemos el sistema iterativamente
for j=1:21
t = j*k;
B_prima = B * w + ff;
w = inv(A) * B_prima;
disp(t, "tiempo")
disp(w, "Sol")
w_sol(1) = cond_izq;
for kk = 1:M
w_sol(kk + 1) = w(kk);
end
w_sol(m) = cond_izq;

```

```
plot(espacio, w_sol);
end
```

2.4.2 Ecuación con Segunda Derivada Temporal

Para el caso de ecuaciones con segunda derivada temporal, esta se aproxima por diferencias finitas centradas en el tiempo, partiendo de la Ec. (2.26), las discretizaciones correspondientes son:

$$u_{tt} \simeq \frac{u_i^{j-1} - 2u_i^j + u_i^{j+1}}{\Delta t^2} \quad (2.32)$$

$$(p(x) u'(x))' \simeq p \left[\frac{u_{i-1}^j - 2u_i^j + u_{i+1}^j}{\Delta x^2} \right] \quad (2.33)$$

$$q(x) u'(x) \simeq q \left[\frac{u_{i-1}^j + u_{i+1}^j}{2\Delta x} \right] \quad (2.34)$$

además de $r(x)u_i^j$ y f_i^j .

Entonces, una vez que se sustituyen las derivadas por su forma en diferencias finitas, lo siguiente es formar el sistema

$$u_i^{j+1} = 2u_i^j - u_i^{j-1} + (\Delta t)^2 B u^j \quad (2.35)$$

esto se logra, colocando del lado izquierdo la igualdad de los términos que contengan el paso del tiempo correspondiente a $j + 1$ y del lado derecho a los correspondientes términos de j y $j - 1$. Para calcular u_i^{j+1} es necesario conocer u_{i-1}, u_i, u_{i+1} en los dos instantes inmediatos anteriores, i.e. t_j y t_{j-1} .

En particular para calcular u_i^1 es necesario conocer u_i^0 y u_i^{-1} , si consideramos

$$u_i^{j+1} = 2u_i^j - u_i^{j-1} + (\Delta t)^2 \mathcal{L}u^j \quad (2.36)$$

para $j = 0$, entonces

$$u_i^1 = 2u_i^0 - u_i^{-1} + (\Delta t)^2 \mathcal{L}u^0 \quad (2.37)$$

de donde tenemos u_i^0 es la condición inicial y $\frac{u_i^1 - u_i^{-1}}{2\Delta t}$ es la primer derivada de la condición inicial, así tenemos que para el primer tiempo tenemos

$$u_i^1 = u(x_i, 0) + \Delta t u'(x_i, 0) + (\Delta t)^2 \mathcal{L}u^0 \quad (2.38)$$

lo cual permite calcular u_i^1 a partir de las condiciones iniciales.

La derivación del método parte de

$$\begin{aligned} u_{tt} &= \mathcal{L}u^j & (2.39) \\ \frac{u_i^{j-1} - 2u_i^j + u_i^{j+1}}{(\Delta t)^2} &= \mathcal{L}u^j \\ u_i^{j+1} &= 2u_i^j - u_i^{j-1} + (\Delta t)^2 \mathcal{L}u^j \end{aligned}$$

donde el error intrínseco a la discretización es de orden cuadrático, pues se ha usado diferencias centradas, tanto para el espacio como para el tiempo.

Ejemplo 6 Sea

$$u_{tt} - 4u''(x) = 0, \quad 0 \leq x \leq l, \quad 0 < t < T$$

sujeta a

$$u(0, t) = u(1, t) = 0, \quad u(x, 0) = \sin(\pi x), \quad u_t(x, 0) = 0$$

con solución analítica

$$u(x, t) = \text{sen}(\pi x) * \text{cos}(2\pi t)$$

entonces el programa queda implementado como:

```
// Dominio
a_ = 0
b_ = 1
// Particion en x
m = 101; // numero de nodos
h = (b_ - a_)/(m-1)
dt = 0.001 // salto del tiempo
// Para que sea estable se debe cumplir que
// sqrt(a) <= h/dt
// Coeficiente
function y = a(x)
y = -4;
endfunction
// Condicion inicial
function y = inicial(x)
y = sin(%pi * x)
endfunction
function y = u_t(x)
y = 0;
endfunction
// Solucion analitica
function y = analitica(x,t)
y = sin(%pi * x) * cos(2* %pi * t)
endfunction
////////////////////////////////////
// Aw^(j+1) = Bw^j
// creo el vector w donde se guardaria la solucion para cada tiempo
// A matriz del lado izquierdo
// B matriz del lado derecho
// B_prima para guardar el resultado de Bw^j
w_sol = zeros(m,1);
w_sol_temp = zeros(m,1);
w_temp = zeros(m,1);
w = zeros(m,1);
```

```

w_ = zeros(m,1);
A = eye(m,m);
B = zeros(m,m);
B_prima = zeros(m,1);
espacio = zeros(m,1)
sol = zeros(m,1);
// primer renglon de cada matriz
B(1,1) = 2*a(a_)*dt*dt/(h*h)
B(1,2) = -a(a_)*dt*dt/(h*h)
// se completa las matrices desde el renglon 2 hasta el m-1
for i = 2:m-1
B(i, i-1) = -a(i*h)*dt*dt/(h*h)
B(i,i) = 2*a(i*h)*dt*dt/(h*h)
B(i,i+1) = -a(i*h)*dt*dt/(h*h)
end
// Ultimo renglon de cada matriz
B(m,m-1) = -a(b_)*dt*dt/(h*h)
B(m,m) = 2*a(b_)*dt*dt/(h*h)
// muestro la matriz
//printf("Matriz B\n");
//disp(B);
for i=1:m
xx = (i-1)*h;
espacio(i) = a_ + xx;
w(i) = inicial(espacio(i)); // Condiciones iniciales
w_(i) = inicial(espacio(i)) + u_t(espacio(i)) * dt
end
//
//disp(espacio)
//disp(w)
//////////
//Para t = 0
B_prima = B * w;
for i = 1:m
w_sol(i) = w_(i) + B_prima(i);
end
//////////
printf("w para t = 0\n");
disp(w_sol);
for i = 1:m
sol(i) = analitica(espacio(i), 0)
end
printf("Solucion analitica para t = 0\n")
disp(sol)
plot(espacio,w_sol)
//plot(espacio,sol,'r')

```

```
w_sol_temp = w_sol;
w_temp = w
for i=1:500
t = i*dt
B_prima = B * w_sol_temp;
w_ = 2 * w_sol_temp - w_temp
w_sol = w_ + B_prima;
////
// for j = 1:m
// sol(j) = analitica(espacio(j), t)
// end
//
// printf("Sol analitica dt = %f", t)
// disp(sol)
// printf("Sol metodo dt = %f", t)
// disp(w_sol)
w_temp = w_sol_temp
w_sol_temp = w_sol
if i == 5 | i == 50 | i == 100 | i == 150 | i == 200 | i == 250 | i ==
300 | i == 350 | i == 400 | i == 450 | i == 500 then
plot(espacio,w_sol)
end
//plot(espacio,sol,'r')
end
```

2.5 Consistencia, Estabilidad, Convergencia y Error del Método de Diferencias Finitas

Cuando se usa algún método para la resolución de ecuaciones diferenciales, se necesita conocer cuan exacta es la aproximación obtenida en comparación con la solución analítica (en caso de existir).

Error Global Sea $U = [U_1, U_2, \dots, U_n]^T$ el vector solución obtenido al utilizar el método de diferencias finitas y $u = [u(x_1), u(x_2), \dots, u(x_n)]$ la solución exacta de los puntos de la malla. El vector de error global se define como $E = U - u$. lo que se desea es que el valor máximo sea pequeño. Usualmente se utilizan distintas normas para encontrar el error.

- La norma infinito, definida como $\|E\|_\infty = \max |e_i|$
- La norma-1, definida como $\|E\|_1 = \sum_{i=1}^n |e_i|$
- La norma-2, definida como $\|E\|_2 = \left(\sum_{i=1}^n e_i^2 \right)^{\frac{1}{2}}$

La norma infinito $\|E\|_\infty$ es en general, la más apropiada para calcular los errores relativos a la discretización.

Definición 7 Si $\|E\| \leq Ch^p, p > 0$, decimos que el método de diferencias finitas es de orden- p de precisión.

Definición 8 Un método de diferencias finitas es llamado convergente si

$$\lim_{h \rightarrow 0} \|E\| = 0. \quad (2.40)$$

Error de Truncamiento Local Sea el operador diferencia $\mathcal{L}u$, definimos el operador en diferencias finitas \mathcal{L}_h , por ejemplo para la ecuación de segundo orden $u''(x) = f(x)$, una de los operadores de diferencias finitas puede ser

$$\mathcal{L}_h u(x) = \frac{u(x-h) - 2u(x) + u(x+h)}{h^2}. \quad (2.41)$$

Definición 9 El error de truncamiento local es definido como

$$T(x) = \mathcal{L}u - \mathcal{L}_h u. \quad (2.42)$$

Para la ecuación diferencial $u''(x) = f(x)$ y el esquema de diferencias centradas usando tres puntos $\frac{u(x-h) - 2u(x) + u(x+h)}{h^2}$, el error de truncamiento local es

$$\begin{aligned} T(x) &= \mathcal{L}u - \mathcal{L}_h u = u''(x) - \frac{u(x-h) - 2u(x) + u(x+h)}{h^2} \\ &= f(x) - \frac{u(x-h) - 2u(x) + u(x+h)}{h^2}. \end{aligned} \quad (2.43)$$

Note que el error de truncamiento local sólo depende de la solución del estencil en diferencias finitas (en el ejemplo es usando tres puntos) pero no depende de la solución global, es por ello que se denomina error de truncamiento local. Este es una medida de que tanto la discretización en diferencias finitas se aproxima a la ecuación diferencial.

Definición 10 Un esquema de diferencias finitas es llamado consistente si

$$\lim_{h \rightarrow 0} T(x) = \lim_{h \rightarrow 0} (\mathcal{L}u - \mathcal{L}_h u) = 0. \quad (2.44)$$

Si $T(x) = Ch^p, p > 0$, entonces se dice que la discretización es de orden- p de precisión, donde C es una constante independiente de h pero puede depender de la solución de $u(x)$. Para conocer cuando un esquema de diferencias finitas es consistente o no, se usa la expansión de Taylor. Por ejemplo, para el esquema de diferencias finitas centradas usando tres puntos para la ecuación diferencial $u''(x) = f(x)$, tenemos que

$$T(x) = u''(x) - \frac{u(x-h) - 2u(x) + u(x+h)}{h^2} = -\frac{h^2}{12} u^{(4)}(x) + \dots = Ch^2 \quad (2.45)$$

donde $C = \frac{1}{12}u^{(4)}(x)$. Por lo tanto, este esquema de diferencias finitas es consistente y la discretización es de segundo orden de precisión.

La consistencia no puede garantizar un esquema de diferencias finitas trabaje. Para ello, necesitamos determinar otra condición para ver si converge o no. Tal condición es llamada la estabilidad de un método de diferencias finitas. Para el problema modelo, tenemos que

$$Au = F + T, \quad AU = F, \quad A(u - U) = T = -AE \quad (2.46)$$

donde A son los coeficientes de la matriz de las ecuaciones en diferencias finitas, F son los términos modificados por la condición de frontera, y T es el vector local de truncamiento en los puntos de la malla.

Si la matriz A es no singular, entonces $\|E\| = \|A^{-1}T\| \leq \|A^{-1}\| \|T\|$. Para el esquema de diferencias finitas centradas, tenemos que $\|E\| = \|A^{-1}\| h^2$. Tal que el error global depende del error de truncamiento local y $\|A^{-1}\|$.

Definición 11 *Un método de diferencias finitas para la ecuación diferencial elíptica es estable si A es invertible y*

$$\|A^{-1}\| \leq C, \text{ para todo } h \leq h_0 \quad (2.47)$$

donde C y h_0 son constantes.

Teorema 12 *Si el método de diferencias finitas es estable y consistente, entonces es convergente.*

3 Consideraciones Sobre la Implementación de Métodos de Solución de Grandes Sistemas de Ecuaciones Lineales

Los modelos matemáticos de muchos sistemas en Ciencia e Ingeniería y en particular una gran cantidad de sistemas continuos geofísicos requieren el procesamiento de sistemas algebraicos de gran escala. En este trabajo se muestra como proceder, para transformar un problema de ecuaciones diferenciales parciales en un sistema algebraico de ecuaciones lineales; y así, poder hallar la solución a dicho problema al resolver el sistema lineal, estos sistemas lineales son expresados en la forma matricial siguiente

$$\underline{\underline{A}}\underline{u} = \underline{f} \quad (3.1)$$

donde la matriz $\underline{\underline{A}}$ es de tamaño $n \times n$ y generalmente bandada, cuyo tamaño de banda es b .

Los métodos de resolución del sistema algebraico de ecuaciones $\underline{\underline{A}}\underline{u} = \underline{f}$ se clasifican en dos grandes grupos (véase [14]): los métodos directos y los métodos iterativos. En los métodos directos la solución \underline{u} se obtiene en un número fijo de pasos y sólo están sujetos a los errores de redondeo. En los métodos iterativos, se realizan iteraciones para aproximarse a la solución \underline{u} aprovechando las características propias de la matriz $\underline{\underline{A}}$, tratando de usar un menor número de pasos que en un método directo (véase [10], [11], [12] y [14]).

Por lo general, es conveniente usar librerías¹⁰ para implementar de forma eficiente a los vectores, matrices —bandadas y dispersas— y resolver los sistemas lineales.

3.1 Métodos Directos

En los métodos directos (véase [10] y [13]), la solución \underline{u} se obtiene en un número fijo de pasos y sólo están sujetos a errores de redondeo. Entre los métodos más importantes se puede considerar: Factorización LU —para matrices simétricas y no simétricas— y Factorización Cholesky —para matrices simétricas—. En todos los casos la matriz original $\underline{\underline{A}}$ es modificada y en caso de usar la Factorización LU el tamaño de la banda b crece a $2b + 1$ si la factorización se realiza en la misma matriz.

3.1.1 Factorización LU

Sea $\underline{\underline{U}}$ una matriz triangular superior obtenida de $\underline{\underline{A}}$ por eliminación bandada. Entonces $\underline{\underline{U}} = \underline{\underline{L}}^{-1}\underline{\underline{A}}$, donde $\underline{\underline{L}}$ es una matriz triangular inferior con unos en la

¹⁰Algunas de las librerías más usadas para resolver sistemas lineales usando matrices bandadas y dispersas son PETCS, HYPRE, ATLAS, LAPACK++, LAPACK, EISPACK, LINPACK, BLAS, entre muchas otras alternativas, tanto para implementaciones secuenciales como paralelas y más recientemente para hacer uso de los procesadores CUDA en las GPU de nVidia.

diagonal. Las entradas de \underline{L}^{-1} pueden obtenerse de los coeficientes \underline{L}_{ij} y pueden ser almacenados estrictamente en las entradas de la diagonal inferior de \underline{A} ya que estas ya fueron eliminadas. Esto proporciona una Factorización \underline{LU} de \underline{A} en la misma matriz \underline{A} ahorrando espacio de memoria, donde el ancho de banda cambia de b a $2b + 1$.

En el algoritmo de Factorización LU, se toma como datos de entrada del sistema $\underline{A}u = \underline{f}$, a la matriz \underline{A} , la cual será factorizada en la misma matriz, esta contendrá a las matrices \underline{L} y \underline{U} producto de la factorización, quedando el método numérico esquemáticamente como:

$$\begin{aligned} &\text{Para } i = 1, 2, \dots, n \{ \\ &\quad \text{Para } j = 1, 2, \dots, n \{ \\ &\quad\quad A_{ji} = A_{ji}/A_{ii} \\ &\quad\quad \text{Para } k = i + 1, \dots, n \{ \\ &\quad\quad\quad A_{jk} = A_{jk} - A_{ji}A_{ik} \\ &\quad\quad\quad \} \\ &\quad\quad \} \\ &\quad \} \end{aligned} \quad (3.2)$$

El problema original $\underline{A}u = \underline{f}$ se escribe como $\underline{LU}u = \underline{f}$, donde la búsqueda de la solución \underline{u} se reduce a la solución sucesiva de los sistemas lineales triangulares

$$\underline{L}y = \underline{f} \quad \text{y} \quad \underline{U}u = \underline{y}. \quad (3.3)$$

Pero recordando que la matriz \underline{A} contiene a las matrices \underline{L} y \underline{U} entonces se tiene que

$$\underline{L}y = \underline{f} \Leftrightarrow \begin{cases} y_1 = f_1/A_{11} \\ y_i = \left(f_i - \sum_{j=1}^{i-1} A_{ij}y_j \right) \text{ para toda } i = 2, \dots, n \end{cases} \quad (3.4)$$

y

$$\underline{U}u = \underline{y} \Leftrightarrow \begin{cases} x_n = y_n/A_{nn} \\ u_i = \frac{1}{A_{ii}} \left(y_i - \sum_{j=i+1}^n A_{ij}x_j \right) \text{ para toda } i = n - 1, \dots, 1 \end{cases} \quad (3.5)$$

La descomposición \underline{LU} requiere¹¹ $n^3/3 - n/3$ multiplicaciones/divisiones y $n^3/3 - n^2/2 + n/6$ sumas/restas —del orden $O(N^3/3)$ operaciones aritméticas

¹¹Notemos que el método de eliminación gaussiana de una matriz \underline{A} de tamaño $n \times n$ requiere $n^3/3 + n^2 - n/3$ multiplicaciones/divisiones además de $n^3/3 + n^2/2 - 5n/6$ sumas/restas. El método de Gauss-Jordan requiere $n^3/2 + n^2 - n/2$ multiplicaciones/divisiones además de $n^3/2 + n/2$ sumas/restas.

para la matriz llena pero sólo del orden $O(Nb^2)$ operaciones aritméticas para la matriz con un ancho de banda de b . La solución de los sistemas $\underline{L}\underline{y} = \underline{f}$ y $\underline{U}\underline{u} = \underline{y}$ requieren $n^2/2 - n/2$ operaciones aritméticas cada uno (véase [10] y [13]).

3.1.2 Factorización Cholesky

Cuando la matriz es simétrica y definida positiva, se obtiene la descomposición \underline{LU} de la matriz $\underline{A} = \underline{LDU} = \underline{LDL}^T$ donde $\underline{D} = \text{diag}(\underline{U})$ es la diagonal con entradas positivas.

En el algoritmo de Factorización Cholesky, se toma como datos de entrada del sistema $\underline{A}\underline{u} = \underline{f}$, a la matriz \underline{A} , la cual será factorizada en la misma matriz y contendrá a la matriz \underline{L} , mientras \underline{L}^T no se calcula, quedando el método numérico esquemáticamente como:

$$\begin{aligned}
 & A_{ii} = \sqrt{A_{11}} \\
 & \text{Para } j = 2, \dots, n \text{ calcule } A_{j1} = A_{j1}/A_{11} \\
 & \text{Para } i = 2, \dots, n \{ \\
 & \quad A_{ii} = \sqrt{A_{ii} - \sum_{k=1}^{i-1} A_{ik}^2} \\
 & \quad \text{Para } j = i + 1, \dots, n \\
 & \quad \quad A_{ji} = \left(A_{ji} - \sum_{k=1}^{i-1} A_{jk}A_{ik} \right) / A_{ii} \\
 & \quad \} \\
 & A_{nn} = \sqrt{A_{nn} - \sum_{k=1}^{n-1} (A_{nk})^2}
 \end{aligned} \tag{3.6}$$

El problema original $\underline{A}\underline{u} = \underline{f}$ se escribe como $\underline{LL}^T\underline{u} = \underline{b}$, donde la búsqueda de la solución \underline{u} se reduce a la solución sucesiva de los sistemas lineales triangulares

$$\underline{L}\underline{y} = \underline{f} \quad \text{y} \quad \underline{L}^T\underline{u} = \underline{y} \tag{3.7}$$

usando la formulación equivalente dada por las Ec.(3.4) y (3.5) para la descomposición LU.

La descomposición \underline{LDL}^T requiere de $n^3/6 + n^2 - 7n/6$ multiplicaciones/divisiones además de $n^3/6 - n/6$ sumas/restas mientras que para la factorización Cholesky se requieren $n^3/6 + n^2/2 - 2n/3$ multiplicaciones/divisiones además de $n^3/6 - n/6$ sumas/restas, adicionalmente se requiere del cálculo de n raíces cuadradas (véase [10] y [13]).

3.1.3 Factorización LU para Matrices Tridiagonales

Como un caso particular de la Factorización LU, está el método de Crout o Thomas y este es aplicable cuando la matriz sólo tiene tres bandas —la diagonal

principal, una antes y una después de la diagonal principal— la Factorización LU se simplifica considerablemente, en este caso las matrices \underline{L} y \underline{U} también se dejan en la matriz \underline{A} y después de la factorización, se resuelven los sistemas $\underline{L}\underline{y} = \underline{f}$ y $\underline{U}\underline{u} = \underline{y}$, quedando el método numérico esquemáticamente como:

$$\begin{aligned}
 A_{12} &= A_{12}/A_{11} \\
 y_1 &= A_{1,n+1}/A_{11} \\
 \text{Para } i &= 2, \dots, n-1 \{ \\
 & \quad A_{ii} = A_{ii} - A_{i,i-1}A_{i-1,i} \\
 & \quad A_{i,i+1} = A_{i,i+1}/A_{ii} \\
 & \quad y_i = (A_{i,n+1} - A_{i,i-1}y_{i-1})/A_{ii} \\
 & \quad \} \\
 A_{nn} &= A_{nn} - A_{n,n-1}A_{n-1,n} \\
 y_n &= (A_{n,n+1} - A_{n,n-1}y_{n-1})/A_{nn} \\
 u_n &= y_n \\
 \text{Para } i &= n-1, \dots, 1 \text{ tome } u_i = y_i - A_{i,i+1}u_{i+1}
 \end{aligned} \tag{3.8}$$

Esta factorización y la resolución del sistema $\underline{A}\underline{u} = \underline{f}$ requiere sólo $(5n-4)$ multiplicaciones/divisiones y $(3n-3)$ sumas/restas, en contraste con la descomposición \underline{LU} que requiere $n^3/3 - n/3$ multiplicaciones/divisiones y $n^3/3 - n^2/2 + n/6$ sumas/restas.

3.2 Métodos Iterativos

En los métodos iterativos, se realizan iteraciones para aproximarse a la solución \underline{u} aprovechando las características propias de la matriz \underline{A} , tratando de usar un menor número de pasos que en un método directo (véase [10] y [13]).

En los métodos iterativos tales como Jacobi, Gauss-Seidel y de Relajación Sucesiva (SOR) en el cual se resuelve el sistema lineal

$$\underline{A}\underline{u} = \underline{f} \tag{3.9}$$

comienza con una aproximación inicial \underline{u}^0 a la solución \underline{u} y genera una sucesión de vectores $\{\underline{u}^k\}_{k=1}^{\infty}$ que converge a \underline{u} . Los métodos iterativos traen consigo un proceso que convierte el sistema $\underline{A}\underline{u} = \underline{f}$ en otro equivalente mediante la iteración de punto fijo de la forma $\underline{u} = \underline{T}\underline{u} + \underline{c}$ para alguna matriz fija \underline{T} y un vector \underline{c} . Luego de seleccionar el vector inicial \underline{u}^0 la sucesión de los vectores de la solución aproximada se genera calculando

$$\underline{u}^k = \underline{T}\underline{u}^{k-1} + \underline{c} \quad \forall k = 1, 2, 3, \dots \tag{3.10}$$

La convergencia a la solución la garantiza el siguiente teorema (véase [14]).

Teorema 13 Si $\|\underline{T}\| < 1$, entonces el sistema lineal $\underline{u} = \underline{T}\underline{u} + \underline{c}$ tiene una solución única \underline{u}^* y las iteraciones \underline{u}^k definidas por la fórmula $\underline{u}^k = \underline{T}\underline{u}^{k-1} + \underline{c} \quad \forall k = 1, 2, 3, \dots$ convergen hacia la solución exacta \underline{u}^* para cualquier aproximación inicial \underline{u}^0 .

Nótese que, mientras menor sea la norma de la matriz \underline{T} , más rápida es la convergencia, en el caso cuando $\|\underline{T}\|$ es menor que uno, pero cercano a uno, la convergencia es lenta y el número de iteraciones necesario para disminuir el error depende significativamente del error inicial. En este caso, es deseable proponer al vector inicial \underline{u}^0 de forma tal que sea mínimo el error inicial. Sin embargo, la elección de dicho vector no tiene importancia si la $\|\underline{T}\|$ es pequeña, ya que la convergencia es rápida.

Como es conocido, la velocidad de convergencia de los métodos iterativos dependen de las propiedades espectrales de la matriz de coeficientes del sistema de ecuaciones, cuando el operador diferencial \mathcal{L} de la ecuación del problema a resolver es auto-adjunto se obtiene una matriz simétrica y positivo definida y el número de condicionamiento de la matriz \underline{A} , es por definición

$$\text{cond}(\underline{A}) = \frac{\lambda_{\max}}{\lambda_{\min}} \geq 1 \quad (3.11)$$

donde λ_{\max} y λ_{\min} es el máximo y mínimo de los eigen-valores de la matriz \underline{A} . Si el número de condicionamiento es cercano a 1 los métodos numéricos al solucionar el problema convergerá en pocas iteraciones, en caso contrario se requerirán muchas iteraciones.

Frecuentemente al usar el método de Elemento Finito, Diferencias Finitas, entre otros, se tiene una velocidad de convergencia de $O\left(\frac{1}{h^2}\right)$ y en el caso de métodos de descomposición de dominio sin preconditionar se tiene una velocidad de convergencia de $O\left(\frac{1}{h}\right)$, donde h es la máxima distancia de separación entre nodos continuos de la partición, es decir, que poseen una pobre velocidad de convergencia cuando $h \rightarrow 0$ (véase [8], [9], [10] y [14]).

Los métodos Jacobi y Gauss-Seidel son usualmente menos eficientes que los métodos discutidos en el resto de esta sección basados en el espacio de Krylov (véase [41] y [13]). Para un ejemplo de este hecho, tomemos el sistema lineal

$$\begin{bmatrix} 4 & 3 & 0 \\ 3 & 4 & -1 \\ 0 & -1 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 24 \\ 30 \\ -24 \end{bmatrix}$$

cuya solución es $x_1 = 3, x_2 = 4, x_3 = -5$; para el método Gauss-Seidel se requirieron 50 iteraciones, Jacobi requirieron 108 iteraciones y Gradiente Conjugado sólo 3 iteraciones.

Los métodos basados en el espacio de Krylov, minimizan en la k -ésima iteración alguna medida de error sobre el espacio afín $\underline{x}_0 + \mathcal{K}_k$, donde \underline{x}_0 es la iteración inicial y \mathcal{K}_k es el k -ésimo subespacio de Krylov

$$\mathcal{K}_k = \text{Generado} \left\{ \underline{r}_0, \underline{A}\underline{r}_0, \dots, \underline{A}^{k-1}\underline{r}_0 \right\} \text{ para } k \geq 1. \quad (3.12)$$

El residual es $\underline{r} = \underline{b} - \underline{A}\underline{x}$, tal $\{\underline{r}_k\}_{k \geq 0}$ denota la sucesión de residuales

$$\underline{r}_k = \underline{b} - \underline{A}\underline{x}_k. \quad (3.13)$$

Entre los métodos más usados definidos en el espacio de Krylov para el tipo de problemas tratados en el presente trabajo se puede considerar: Método de Gradiente Conjugado —para matrices simétricas— y GMRES —para matrices no simétricas—.

3.2.1 Método de Gradiente Conjugado

Si la matriz generada por la discretización es simétrica — $\underline{A} = \underline{A}^T$ — y definida positiva — $\underline{u}^T \underline{A} \underline{u} > 0$ para todo $\underline{u} \neq 0$ —, entonces es aplicable el método de Gradiente Conjugado —Conjugate Gradient Method (CGM)—. La idea básica en que descansa el método del Gradiente Conjugado consiste en construir una base de vectores ortogonales espacio de Krylov $\mathcal{K}_n(\underline{A}, \underline{v}^n)$ y utilizarla para realizar la búsqueda de la solución en forma lo más eficiente posible.

Tal forma de proceder generalmente no sería aconsejable porqué la construcción de una base ortogonal utilizando el procedimiento de Gram-Schmidt requiere, al seleccionar cada nuevo elemento de la base, asegurar su ortogonalidad con respecto a cada uno de los vectores construidos previamente. La gran ventaja del método de Gradiente Conjugado radica en que cuando se utiliza este procedimiento, basta con asegurar la ortogonalidad de un nuevo miembro con respecto al último que se ha construido, para que automáticamente esta condición se cumpla con respecto a todos los anteriores.

En el algoritmo de Gradiente Conjugado, se toma a la matriz \underline{A} como simétrica y positiva definida, y como datos de entrada del sistema $\underline{A} \underline{u} = \underline{f}$, el vector de búsqueda inicial \underline{u}^0 y se calcula $\underline{r}^0 = \underline{f} - \underline{A} \underline{u}^0$, $\underline{p}^0 = \underline{r}^0$, quedando el método numérico esquemáticamente como:

$$\begin{aligned}
 \alpha^n &= \frac{\langle \underline{p}^n, \underline{p}^n \rangle}{\langle \underline{p}^n, \underline{A} \underline{p}^n \rangle} \\
 \underline{u}^{n+1} &= \underline{u}^n + \alpha^n \underline{p}^n \\
 \underline{r}^{n+1} &= \underline{r}^n - \alpha^n \underline{A} \underline{p}^n \\
 &\text{Prueba de convergencia} \\
 \beta^n &= \frac{\langle \underline{r}^{n+1}, \underline{r}^{n+1} \rangle}{\langle \underline{r}^n, \underline{r}^n \rangle} \\
 \underline{p}^{n+1} &= \underline{r}^{n+1} + \beta^n \underline{p}^n \\
 n &= n + 1
 \end{aligned} \tag{3.14}$$

donde $\langle \cdot, \cdot \rangle = (\cdot, \cdot)$ será el producto interior adecuado al sistema lineal en particular, la solución aproximada será \underline{u}^{n+1} y el vector residual será \underline{r}^{n+1} .

En la implementación numérica y computacional del método es necesario realizar la menor cantidad de operaciones posibles por iteración, en particular en $\underline{A} \underline{p}^n$, una manera de hacerlo queda esquemáticamente como:

Dado el vector de búsqueda inicial \underline{u} , calcula $\underline{r} = \underline{f} - \underline{A} \underline{u}$, $\underline{p} = \underline{r}$ y $\mu = \underline{r} \cdot \underline{r}$.

Para $n = 1, 2, \dots$, Mientras $(\mu < \varepsilon)$ {

$$\begin{aligned}
 v &= \underline{A} \underline{p} \\
 \alpha &= \frac{\underline{r} \cdot v}{\underline{p} \cdot v} \\
 \underline{u} &= \underline{u} + \alpha \underline{p}
 \end{aligned}$$

$$\left. \begin{aligned} \underline{r} &= \underline{r} - \alpha \underline{v} \\ \mu' &= \underline{r} \cdot \underline{r} \\ \beta &= \frac{\mu'}{\mu} \\ \underline{p} &= \underline{r} + \beta \underline{p} \\ \mu &= \mu' \end{aligned} \right\}$$

La solución aproximada será \underline{u} y el vector residual será \underline{r} .

Si se denota con $\{\lambda_i, V_i\}_{i=1}^N$ las eigen-soluciones de \underline{A} , i.e. $\underline{A}V_i = \lambda_i V_i$, $i = 0, 1, 2, \dots, N$. Ya que la matriz \underline{A} es simétrica, los eigen-valores son reales y se pueden ordenar $\lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_N$. Se define el número de condición por $Cond(\underline{A}) = \lambda_N/\lambda_1$ y la norma de la energía asociada a \underline{A} por $\|\underline{u}\|_{\underline{A}}^2 = \underline{u} \cdot \underline{A}u$ entonces

$$\|\underline{u} - \underline{u}^k\|_{\underline{A}} \leq \|\underline{u} - \underline{u}^0\|_{\underline{A}} \left[\frac{1 - \sqrt{Cond(\underline{A})}}{1 + \sqrt{Cond(\underline{A})}} \right]^{2k}. \quad (3.15)$$

El siguiente teorema da idea del espectro de convergencia del sistema $\underline{A}u = \underline{b}$ para el método de Gradiente Conjugado.

Teorema 14 Sea $\kappa = cond(\underline{A}) = \frac{\lambda_{\max}}{\lambda_{\min}} \geq 1$, entonces el método de Gradiente Conjugado satisface la \underline{A} -norma del error dado por

$$\frac{\|\underline{e}^n\|}{\|\underline{e}^0\|} \leq \frac{2}{\left[\left(\frac{\sqrt{\kappa}+1}{\sqrt{\kappa}-1} \right)^n + \left(\frac{\sqrt{\kappa}+1}{\sqrt{\kappa}-1} \right)^{-n} \right]} \leq 2 \left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1} \right)^n \quad (3.16)$$

donde $\underline{e}^m = \underline{u} - \underline{u}^m$ del sistema $\underline{A}u = \underline{b}$.

Nótese que para κ grande se tiene que

$$\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1} \simeq 1 - \frac{2}{\sqrt{\kappa}} \quad (3.17)$$

tal que

$$\|\underline{e}^n\|_{\underline{A}} \simeq \|\underline{e}^0\|_{\underline{A}} \exp\left(-2\frac{n}{\sqrt{\kappa}}\right) \quad (3.18)$$

de lo anterior se puede esperar un espectro de convergencia del orden de $O(\sqrt{\kappa})$ iteraciones (véase [14] y [41]).

Definición 15 Un método iterativo para la solución de un sistema lineal es llamado óptimo, si la razón de convergencia a la solución exacta es independiente del tamaño del sistema lineal.

Definición 16 Un método para la solución del sistema lineal generado por métodos de descomposición de dominio es llamado escalable, si la razón de convergencia no se deteriora cuando el número de subdominios crece.

3.2.2 Método Residual Mínimo Generalizado

Si la matriz generada por la discretización es no simétrica, entonces una opción, es el método Residual Mínimo Generalizado —Generalized Minimum Residual Method (GMRES)—, este representa una formulación iterativa común satisfaciendo una condición de optimización. La idea básica detrás del método se basa en construir una base ortonormal

$$\{\underline{v}^1, \underline{v}^2, \dots, \underline{v}^n\} \quad (3.19)$$

para el espacio de Krylov $\mathcal{K}_n(\underline{A}, \underline{v}^n)$. Para hacer \underline{v}^{n+1} ortogonal a $\mathcal{K}_n(\underline{A}, \underline{v}^n)$, es necesario usar todos los vectores previamente construidos $\{\underline{v}^{n+1j}\}_{j=1}^n$ —en la práctica sólo se guardan algunos vectores anteriores— en los cálculos. Y el algoritmo se basa en una modificación del método de Gram-Schmidt para la generación de una base ortonormal. Sea $\underline{V}_n = [\underline{v}^1, \underline{v}^2, \dots, \underline{v}^n]$ la cual denota la matriz conteniendo \underline{v}^j en la j -ésima columna, para $j = 1, 2, \dots, n$, y sea $\underline{H}_n = [h_{i,j}]$, $1 \leq i, j \leq n$, donde las entradas de \underline{H}_n no especificadas en el algoritmo son cero. Entonces, \underline{H}_n es una matriz superior de Hessenberg. i.e. $h_{ij} = 0$ para $j < i - 1$, y

$$\begin{aligned} \underline{AV}_n &= \underline{V}_n \underline{H}_n + h_{n+1,n} [0, \dots, 0, \underline{v}^{n+1}] \\ \underline{H}_n &= \underline{H}_n^T \underline{AV}_n. \end{aligned} \quad (3.20)$$

En el algoritmo del método Residual Mínimo Generalizado, la matriz \underline{A} es tomada como no simétrica, y como datos de entrada del sistema

$$\underline{Au} = \underline{f} \quad (3.21)$$

el vector de búsqueda inicial \underline{u}^0 y se calcula $\underline{r}^0 = \underline{f} - \underline{Au}^0$, $\beta^0 = \|\underline{r}^0\|$, $\underline{v}^1 = \underline{r}^0/\beta^0$, quedando el método esquemáticamente como:

$$\begin{aligned} &\text{Para } n = 1, 2, \dots, \text{ Mientras } \beta^n < \tau\beta^0 \{ \\ &\quad \underline{w}_0^{n+1} = \underline{Av}^n \\ &\quad \text{Para } l = 1 \text{ hasta } n \{ \\ &\quad\quad h_{l,n} = \langle \underline{w}_l^{n+1}, \underline{v}^l \rangle \\ &\quad\quad \underline{w}_{l+1}^{n+1} = \underline{w}_l^{n+1} - h_{l,n} \underline{v}^l \\ &\quad\quad \} \\ &\quad h_{n+1,n} = \|\underline{w}_{n+1}^{n+1}\| \\ &\quad \underline{v}^{n+1} = \underline{w}_{n+1}^{n+1}/h_{n+1,n} \\ &\quad \text{Calcular } \underline{y}^n \text{ tal que } \beta^n = \|\beta^0 \underline{e}_1 - \hat{\underline{H}}_n \underline{y}^n\| \text{ es mínima} \\ &\quad \} \end{aligned} \quad (3.22)$$

donde $\hat{\underline{H}}_n = [h_{ij}]_{1 \leq i \leq n+1, 1 \leq j \leq n}$, la solución aproximada será $\underline{u}^n = \underline{u}^0 + \underline{V}_n \underline{y}^n$, y el vector residual será

$$\underline{r}^n = \underline{r}^0 - \underline{AV}_n \underline{y}^n = \underline{V}_{n+1} (\beta^0 \underline{e}_1 - \hat{\underline{H}}_n \underline{y}^n). \quad (3.23)$$

Teorema 17 Sea \underline{u}^k la iteración generada después de k iteraciones de GMRES, con residual \underline{r}^k . Si la matriz \underline{A} es diagonalizable, i.e. $\underline{A} = \underline{V}\underline{\Lambda}\underline{V}^{-1}$ donde $\underline{\Lambda}$ es una matriz diagonal de eigen-valores de \underline{A} , y \underline{V} es la matriz cuyas columnas son los eigen-vectores, entonces

$$\frac{\|\underline{r}^k\|}{\|\underline{r}^0\|} \leq \kappa(V) \min_{p_\kappa \in \Pi_\kappa, p_\kappa(0)=1} \max_{\lambda_j} |p_\kappa(\lambda_j)| \quad (3.24)$$

donde $\kappa(V) = \frac{\|\underline{V}\|}{\|\underline{V}^{-1}\|}$ es el número de condicionamiento de \underline{V} .

3.3 Estructura Óptima de las Matrices en su Implementación Computacional

Una parte fundamental de la implementación computacional de los métodos numéricos de resolución de sistemas algebraicos, es utilizar una forma óptima de almacenar¹², recuperar y operar las matrices, tal que, facilite los cálculos que involucra la resolución de grandes sistemas de ecuaciones lineales cuya implementación puede ser secuencial o paralela (véase [13]).

El sistema lineal puede ser expresado en la forma matricial $\underline{A}\underline{u} = \underline{f}$, donde la matriz \underline{A} —que puede ser real o virtual— es de tamaño $n \times n$ con banda b , pero el número total de datos almacenados en ella es a los más $n*b$ números de doble precisión, en el caso de ser simétrica la matriz, el número de datos almacenados es menor a $(n*b)/2$. Además si el problema que la originó es de coeficientes constantes el número de valores almacenados se reduce drásticamente a sólo el tamaño de la banda b .

En el caso de que el método para la resolución del sistema lineal a usar sea del tipo Factorización LU o Cholesky, la estructura de la matriz cambia, ampliándose el tamaño de la banda de b a $2*b+1$ en la factorización, en el caso de usar métodos iterativos tipo CGM o GMRES la matriz se mantiene intacta con una banda b .

Para la resolución del sistema lineal virtual asociada a los métodos de diferencias finitas, la operación básica que se realiza de manera reiterada, es la multiplicación de una matriz por un vector $\underline{v} = \underline{C}\underline{u}$, la cual es necesario realizar de la forma más eficiente posible.

Un factor determinante en la implementación computacional, para que esta resulte eficiente, es la forma de almacenar, recuperar y realizar las operaciones que involucren matrices y vectores, de tal forma que la multiplicación se realice en la menor cantidad de operaciones y que los valores necesarios para realizar dichas operaciones queden en la medida de lo posible contiguos para ser almacenados en el Cache¹³ del procesador.

¹²En el caso de los ejemplos de la sección (4) de MatLab y C++ se usan matrices que minimizan la memoria usada en el almacenamiento de las matrices y maximizan la eficiencia de los métodos numéricos de solución del sistema lineal asociado.

¹³Nótese que la velocidad de acceso a la memoria principal (RAM) es relativamente lenta con respecto al Cache, este generalmente está dividido en sub-Caches L1 —de menor tamaño y

Dado que la multiplicación de una matriz \underline{C} por un vector \underline{u} , dejando el resultado en \underline{v} se realiza mediante el algoritmo

```
for (i=0; i<ren; i++)
{
    s = 0.0;
    for (j=0; j < col; j++)
    {
        s += C[i][j]*u[j];
    }
    v[i] = s;
}
```

Para lograr una eficiente implementación del algoritmo anterior, es necesario que el gran volumen de datos desplazados de la memoria al Cache y viceversa sea mínimo. Por ello, los datos se deben agrupar para que la operación más usada —en este caso multiplicación matriz por vector— se realice con la menor solicitud de datos a la memoria principal, si los datos usados —renglón de la matriz— se ponen contiguos minimizará los accesos a la memoria principal, pues es más probable que estos estarán contiguos en el Cache al momento de realizar la multiplicación.

Por ejemplo, en el caso de matrices bandadas de tamaño de banda b , el algoritmo anterior se simplifica a

```
for (i=0; i<ren; i++)
{
    s= 0.0;
    for (k=0; k < ban; k++)
    {
        if ((Ind[k] + i) >= 0 && (Ind[k]+i) < ren)
            s += Dat[i][k]*u[Ind[k]+i];
    }
    v[i]=s;
}
```

Si, la solicitud de memoria para $\text{Dat}[i]$ se hace de tal forma que los datos del renglón estén continuos —son b números de punto flotante—, esto minimizará los accesos a la memoria principal en cada una de las operaciones involucradas en el producto, como se explica en las siguientes secciones.

el más rápido—, L2 y hasta L3 —el más lento y de mayor tamaño— los cuales son de tamaño muy reducido con respecto a la RAM.

Por ello, cada vez que las unidades funcionales de la Unidad de Aritmética y Lógica requieren un conjunto de datos para implementar una determinada operación en los registros, solicitan los datos primeramente a los Caches, estos consumen diversa cantidad de ciclos de reloj para entregar el dato si lo tienen —pero siempre el tiempo es menor que solicitarle el dato a la memoria principal—; en caso de no tenerlo, se solicitan a la RAM para ser cargados a los caches y poder implementar la operación solicitada.

3.3.1 Matrices Bandadas

En el caso de las matrices bandadas de banda b —sin pérdida de generalidad y para propósitos de ejemplificación se supone pentadiagonal— típicamente tiene la siguiente forma

$$\underline{\underline{A}} = \begin{bmatrix} a_1 & b_1 & & & c_1 & & & & \\ d_2 & a_2 & b_2 & & & c_2 & & & \\ & d_3 & a_3 & b_3 & & & c_3 & & \\ & & d_4 & a_4 & b_4 & & & c_4 & \\ e_5 & & & d_5 & a_5 & b_5 & & & c_5 \\ & e_6 & & & d_6 & a_6 & b_6 & & \\ & & e_7 & & & d_7 & a_7 & b_7 & \\ & & & e_8 & & & d_8 & a_8 & b_8 \\ & & & & e_9 & & & d_9 & a_9 \end{bmatrix} \quad (3.25)$$

la cual puede ser almacenada usando el algoritmo (véase [13]) Compressed Diagonal Storage (CDS), optimizado para ser usado en C++, para su almacenamiento y posterior recuperación. Para este ejemplo en particular, se hará uso de un vector de índices

$$\underline{\underline{Ind}} = [-5, -1, 0, +1, +5] \quad (3.26)$$

y los datos serán almacenados usando la estructura

$$\underline{\underline{Dat}} = \begin{bmatrix} 0 & 0 & a_1 & b_1 & c_1 \\ 0 & d_2 & a_2 & b_2 & c_2 \\ 0 & d_3 & a_3 & b_3 & c_3 \\ 0 & d_4 & a_4 & b_4 & c_4 \\ e_5 & d_5 & a_5 & b_5 & c_5 \\ e_6 & d_6 & a_6 & b_6 & 0 \\ e_7 & d_7 & a_7 & b_7 & 0 \\ e_8 & d_8 & a_8 & b_8 & 0 \\ e_9 & d_9 & a_9 & 0 & 0 \end{bmatrix} \quad (3.27)$$

de tal forma que la matriz $\underline{\underline{A}}$ puede ser reconstruida de forma eficiente. Para obtener el valor $A_{i,j}$, calculo $ind = j - i$, si el valor ind esta en la lista de índices $\underline{\underline{Ind}}$ —supóngase en la columna k —, entonces $A_{i,j} = \underline{\underline{Dat}}_{ik}$, en otro caso $A_{i,j} = 0$.

Casos Particulares de la Matriz Bandada $\underline{\underline{A}}$ Básicamente dos casos particulares surgen en el tratamiento de ecuaciones diferenciales parciales: El primer caso es cuando el operador diferencial parcial es simétrico y el otro, en el que los coeficientes del operador sean constantes.

Para el primer caso, al ser la matriz simétrica, sólo es necesario almacenar la parte con índices mayores o iguales a cero, de tal forma que se buscara el índice que satisfaga $ind = |j - i|$, reduciendo el tamaño de la banda a $b/2$ en la matriz $\underline{\underline{A}}$.

Para el segundo caso, al tener coeficientes constantes el operador diferencial, los valores de los renglones dentro de cada columna de la matriz son iguales, y sólo es necesario almacenarlos una sola vez, reduciendo drásticamente el tamaño de la matriz de datos.

3.3.2 Matrices Dispersas

Las matrices dispersas de a lo más b valores distintos por renglón —sin pérdida de generalidad y para propósitos de ejemplificación se supone $b = 3$ — que surgen en métodos de descomposición de dominio para almacenar algunas matrices, típicamente tienen la siguiente forma

$$\underline{\underline{A}} = \begin{bmatrix} a_1 & & & & b_1 & & & & c_1 \\ & a_2 & & & b_2 & & & & c_2 \\ & & & & a_3 & & b_3 & & c_3 \\ a_4 & & & b_4 & & & & & \\ & & a_5 & & b_5 & & & & c_5 \\ a_6 & b_6 & c_6 & & & & & & \\ & & & & & & a_7 & b_7 & c_7 \\ & & & a_8 & & & b_8 & c_8 & \\ & & & & & & a_9 & b_9 & \end{bmatrix} \quad (3.28)$$

la cual puede ser almacenada usando el algoritmo (véase [13]) Jagged Diagonal Storage (JDC), optimizado para ser usado en C++. Para este ejemplo en particular, se hará uso de una matriz de índices

$$\underline{\underline{Ind}} = \begin{bmatrix} 1 & 6 & 9 \\ 2 & 5 & 8 \\ 5 & 8 & 9 \\ 1 & 4 & 0 \\ 3 & 6 & 9 \\ 1 & 2 & 3 \\ 7 & 8 & 9 \\ 4 & 7 & 8 \\ 7 & 8 & 0 \end{bmatrix} \quad (3.29)$$

y los datos serán almacenados usando la estructura

$$\underline{\underline{Dat}} = \begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \\ a_4 & b_4 & 0 \\ a_5 & b_5 & c_5 \\ a_6 & b_6 & c_6 \\ a_7 & b_7 & c_7 \\ a_8 & b_8 & c_8 \\ a_9 & b_9 & 0 \end{bmatrix} \quad (3.30)$$

de tal forma que la matriz \underline{A} puede ser reconstruida de forma eficiente. Para obtener el valor $A_{i,j}$, busco el valor j en la lista de índices \underline{Ind} dentro del renglón i , si lo encuentro en la posición k , entonces $A_{i,j} = \text{Dat}_{ik}$, en otro caso $A_{i,j} = 0$.

Casos Particulares de la Matriz Dispersa \underline{A} Si la matriz \underline{A} , que al ser almacenada, se observa que existen a lo más r diferentes renglones con valores distintos de los n con que cuenta la matriz y si $r \ll n$, entonces es posible sólo guardar los r renglones distintos y llevar un arreglo que contenga la referencia al renglón almacenado.

3.3.3 Multiplicación Matriz-Vector

Los métodos de descomposición de dominio requieren por un lado la resolución de al menos un sistema lineal y por el otro lado requieren realizar la operación de multiplicación de matriz por vector, i.e. $\underline{C}\underline{u}$ de la forma más eficiente posible, por ello los datos se almacenan de tal forma que la multiplicación se realice en la menor cantidad de operaciones.

Dado que la multiplicación de una matriz \underline{C} por un vector \underline{u} , dejando el resultado en \underline{v} se realiza mediante el algoritmo:

```
for (i=0; i<ren; i++)
{
    s = 0.0;
    for (j=0; j < col; j++)
    {
        s += C[i][j]*u[j];
    }
    v[i] = s;
}
```

En el caso de matrices bandadas, se simplifica a:

```
for (i=0; i<ren; i++)
{
    s = 0.0;
    for (k=0; k < ban; k++)
    {
        if ((Ind[k] + i) >= 0 && (Ind[k]+i) < ren)
            s += Dat[i][k]*u[Ind[k]+i];
    }
    v[i]=s;
}
```

De forma similar, en el caso de matrices dispersas, se simplifica a:

```
for (i=0; i<ren; i++)
{
  s = 0.0, k = 0
  while (Ind[i][k] != -1)
  {
    s += Dat[i][k]*u[Ind[i][k]];
    k++;
    if (k >= b) break;
  }
  v[i] = s;
}
```

De esta forma, al tomar en cuenta la operación de multiplicación de una matriz por un vector, donde el renglón de la matriz involucrado en la multiplicación queda generalmente en una región contigua del Cache, se hace óptima la operación de multiplicación de matriz por vector.

4 Implementación Computacional del Método de Diferencias Finitas para la Resolución de Ecuaciones Diferenciales Parciales

Existen diferentes paquetes y lenguajes de programación en los cuales se puede implementar eficientemente la solución numérica de ecuaciones diferenciales parciales mediante el método de diferencias Finitas, en este capítulo se describe la implementación¹⁴ en los paquetes de cómputo SciLab y Octave (MatLab) y en el lenguaje de programación C++ y Python.

4.1 Implementación en SciLab

Scilab¹⁵ es un paquete de cómputo open source para el cálculo numérico el cual provee un poderoso ambiente de cálculo para aplicaciones Científicas y de Ingeniería.

Ejemplo 18 *Sea*

$$u''(x) = -\pi^2 \cos(\pi x), \quad 0 \leq x \leq 0.5, \quad u(0) = 1, \quad u'(0.5) = -\pi$$

entonces el programa queda implementado como:

```
function y=LadoDerecho(x)
y=-%pi*%pi*cos(%pi*x);
endfunction
```

```
function y=SolucionAnalitica(x)
y=cos(%pi*x);
endfunction
```

```
a=0; // Inicio dominio
c=0.5; // Fin dominio
M=40; // Partición
N=M-1; // Nodos interiores
h=(c-a)/(M-1); // Incremento en la malla
Y0=1; // Condición Dirchlet inicial en el inicio del dominio
Y1=-%pi; // Condición Neumann inicial en el fin del dominio
A=zeros(N,N); // Matriz A
b=zeros(N); // Vector b

R=1/(h^2);
P=-2/(h^2);
Q=1/(h^2);
```

¹⁴ En el caso de los ejemplos de MatLab, C++ y Python se usan matrices que minimizan la memoria usada en el almacenamiento de las matrices y maximizan la eficiencia de los métodos numéricos de solución del sistema lineal asociado.

¹⁵ Scilab [<http://www.scilab.org>]


```
// Primer renglon de la matriz A y vector b
A(1,1)=P;
A(1,2)=Q;
b(1)=LadoDerecho(a)-Y0*R; // Frontera Dirichlet
// Renglones intermedios de la matriz A y vector b
for i=2:N-1
    A(i,i-1)=R;
    A(i,i)=P;
    A(i,i+1)=Q;
    b(i)=LadoDerecho(a+h*(i-1));
end
// Renglón final de la matriz A y vector b
A(N,N-1)=-1/(h^2);
A(N,N)=-1/(h^2);
b(N)=Y1/h; // Frontera Neumann

// Resuelve el sistema lineal Ax=b
x=inv(A)*b;

// Prepara la graficación
xx=zeros(M,1);
zz=zeros(M,1);
for i=1:M
    xx(i)=a+h*(i-1);
    zz(i)=SolucionAnalitica(xx(i));
end
yy=zeros(M,1);
yy(1)=Y0; // Condición inicial
for i=1:N
    yy(i+1)=x(i);
end
// Grafica la solución de la Ecuación Diferencial Parcial en 1D
plot2d(xx,[yy,zz])
```

4.2 Implementación en Octave (MatLab)

GNU Octave¹⁶ es un paquete de cómputo open source para el cálculo numérico —muy parecido a MatLab¹⁷ pero sin costo alguno para el usuario— el cual provee un poderoso ambiente de cálculo para aplicaciones Científicas y de Ingeniería.

Ejemplo 19 *Sea*

$$-u''(x) = -\pi^2 \cos(\pi x), \quad 0 \leq x \leq 1, \quad u(0) = 1, \quad u(1) = -1$$

entonces el programa queda implementado como:

```
% Método de diferencias Finitas una Dimensión
function [x,U] = fdm1d(a,b,ua,ub,n)
h = (b-a)/n;
h1 = h*h;
A = sparse(n-1,n-1)
F = zeros(n-1,1)
for i=1:n-2,
A(i,i) = -2/h1;
A(i+1,i) = 1/h1;
A(i,i+1) = 1/h1;
end
A(n-1,n-1) = -2/h1;
for i=1:n-1,
x(i) = a + i*h;
F(i) = f(x(i));
end
F(1) = F(1) - ua/h1;
F(n-1) = F(n-1) - ub/h1;

% Soluciona el sistema
U = A\F;

% Graficar la solución numérica
plot(x,U,'o'); hold
% Graficar la solución analítica
u = zeros(n-1,1);
for i=1: n-1,
u(i) = cos(pi*x(i));
end
plot(x,u)

% Grafica el error
figure(2);
```

¹⁶GNU Octave [<http://www.gnu.org/software/octave/>]

¹⁷MatLab [<http://www.mathworks.com/products/matlab/>]

```
plot(x,U-u);  
return  
  
% Lado derecho  
function y = f(x)  
y = -pi*pi*cos(pi*x);  
return
```

Para hacer uso del programa, escribir en la consola

```
[x,U] = fdm1d(0,1,1,-1,40);
```

4.3 Implementación en C++

GMM++¹⁸ es una librería para C++ que permite definir diversos tipos de matrices y vectores además operaciones básicas de algebra lineal. La facilidad de uso y la gran cantidad de opciones hacen que GMM++ sea una buena opción para trabajar con operaciones elementales de algebra lineal.

Se instala en Debian Linux y/o Ubuntu como:

```
# apt-get install libgmm++-dev
```

Para compilar el ejemplo usar:

```
$ g++ ejemplo.cpp
```

Para ejecutar usar:

```
$ ./a.out
```

Ejemplo 20 Sea

$$-u''(x) + u(x) = 0, \quad 0 \leq x \leq 1, \quad u(0) = 0, \quad u(1) = 1$$

entonces el programa queda implementado como:

```
#include <gmm/gmm.h>
int main(void)
{
  int i;
  double a=0; // Inicio dominio
  double c=1; // Fin dominio
  int M=20; // Partición
  int N=M-2; // Nodos interiores
  double h=(c-a)/(M-1); // Incremento en la malla
  double Y0=0.0; // Condicion inicial en el inicio del dominio
  double Y1=1.0; // Condicion inicial en el fin del dominio

  // Matriz densa
  gmm::dense_matrix<double> AA(N, N);

  // Matriz dispersa
  gmm::row_matrix< gmm::rsvector<double> > A(N, N);

  // Vectores
  std::vector<double> x(N), b(N);

  double P=2/(h*h);
  double Q=-1/(h*h)+1/(2*h);
  double R=-1/(h*h)-1/(2*h);
  // Primer renglon de la matriz A y vector b
  A(0,0)=P;
  A(0,1)=Q;
```

¹⁸GMM++ [<http://download.gna.org/getfem/html/homepage/gmm/>]

```
b[0]=-Y0*R;
// Renglones intermedios de la matriz A y vector b
for(i=1;i <N-1;i++)
{
A(i,i-1)=R;
A(i,i)=P;
A(i,i+1)=Q;
}
// Renglón final de la matriz A y vector b
A(N-1,N-2)=R;
A(N-1,N-1)=P;
b[N-1]=-Y1*Q;

gmm::copy(A,AA);
// LU para matrices densa
gmm::lu_solve(AA, x, b);
std::cout << "LU"<< x << gmm::endl;

gmm::identity_matrix PS; // Optional scalar product for cg
gmm::identity_matrix PR; // Optional preconditioner
gmm::iteration_iter(10E-6); // Iteration object with the max residu
size_t restart = 50; // restart parameter for GMRES
gmm::cg(A, x, b, PS, PR, iter); // Conjugate gradient

std::cout << "CGM"<< x << std::endl;
gmm::bicgstab(A, x, b, PR, iter); // BICGSTAB BiConjugate Gradient Sta-
bilized

std::cout << "BICGSTAB"<< x << std::endl;
gmm::gmres(A, x, b, PR, restart, iter); // GMRES generalized minimum
residual

std::cout << "GMRES"<< x << std::endl;
gmm::qmr(A, x, b, PR, iter); // Quasi-Minimal Residual method.
std::cout << "Quasi-Minimal"<< x << std::endl;

// computation of a preconditioner (ILUT)
gmm::ilut_precond< gmm::row_matrix< gmm::rsvector<double> > > Pre(A,
10, 1e-4);
gmm::gmres(A, x, b, Pre, restart, iter); // execute the GMRES algorithm
std::cout << "GMRES preconditiones ILUT"<< x << std::endl;
return 0;
}
```

4.4 Implementación en Python

Python es un lenguaje de programación interpretado multiparadigma cuya filosofía hace hincapie en una sintaxis que favorezca un código legible, que corre en multiplataforma y posee una licencia de código abierto, el cual provee un poderoso ambiente de cálculo para aplicaciones Científicas y de Ingeniería.

Ejemplo 21 Sea

$$u_t + au(x) = 0, \quad -2 \leq x \leq 8$$

mediante el esquema Upwind, entonces el programa queda implementado como:

```
# -*- coding: utf-8 -*-
"""
    @author: omar jonathan mendoza bernal
    Upwind scheme for
    u_t + a(x) u_x = 0
    Ejemplo
    u_t + a u_x = 0
    Si u(x,0) = f(x) es la condición inicial
    la solución analítica es
    u(x,t) = f(x-at)
    """

from math import exp
from scipy import sparse
import numpy as np
import matplotlib.pyplot as plt
# Ecuación
# u_t + 2 u_x = 0
coef_a = 2
def condicion_inicial(x):
    """
    Condición inicial de la ecuación
    """
    y = exp(-(x - 0.2)*(x - 0.02))
    return y
def sol_analitica(x, t):
    """
    Solución analítica de la ecuación
    """
    y = exp(-(x-2*t)*(x-2*t))
    return y
#####
# Dominio
#####
a = -2.0
b = 8.0
# Partición del dominio
```

```
nNodos = 401
h = (b - a)/(nNodos - 1)
# Intervalo de tiempo
dt = 0.012
# creo el vector w donde se guardará la solución para cada tiempo
# B matriz del lado derecho
w = np.zeros((nNodos,1))
B = np.zeros((nNodos, nNodos))
B = np.matrix(B)
espacio = np.zeros((nNodos,1))
for i in xrange(nNodos):
    xx_ = a + i*h
    espacio[i] = xx_
    w[i] = condicion_inicial(xx_)
    print "Espacio"
    print espacio
    print "Condición Inicial"
    print w
    mu = coef_a * dt / h
    if mu <= 1:
        print "mu ", mu
        print "Buena aproximación"
    else:
        print "mu ", mu
        print "Mala aproximación"
    if coef_a >= 0:
        B[0,0] = 1 - mu
        for i in xrange(1, nNodos):
            B[i,i-1] = mu
            B[i,i] = 1 - mu
        else:
            B[0,0] = 1 + mu;
            B[0,1] = -mu;
            # se completa las matrices desde el renglon 2 hasta el m-2
            for i in xrange(1, nNodos-1):
                B[i,i] = 1 + mu;
                B[i, i+1] = -mu;
            B[nNodos-1,nNodos-1] = 1 + mu
            # para guardar la solución analítica
            xx = np.zeros((nNodos,1));
            iteraciones = 201
            # Matriz sparse csr
            Bs = sparse.csr_matrix(B);
            # Resolvemos el sistema iterativamente
            for j in xrange(1, iteraciones):
                t = j*dt;
```

```
w = Bs*w;  
    # Imprimir cada 20 iteraciones  
if j%20 == 0:  
    print "t", t  
    print "w", w  
    plt.plot(espacio, w)
```


5 Bibliografía

Referencias

- [1] K. Hutter y K Jöhnk, *Continuum Methods of Physical Modeling*, Springer-Verlag Berlin Heidelberg New York, 2004.
- [2] J. L. Lions y E. Magenes, *Non-Homogeneous Boundary Value Problems and Applications* Vol. I, Springer-Verlag Berlin Heidelberg New York, 1972.
- [3] A. Quarteroni y A. Valli, *Domain Decomposition Methods for Partial Differential Equations*. Clarendon Press Oxford, 1999.
- [4] A. Quarteroni y A. Valli; *Numerical Approximation of Partial Differential Equations*. Springer, 1994.
- [5] B. Dietrich, *Finite Elements: Theory, Fast Solvers, and Applications in Solid Mechanics*, Cambridge University, 2001.
- [6] B. F. Smith, P. E. Bjørstad, W. D. Gropp; *Domain Decomposition, Parallel Multilevel Methods for Elliptic Partial Differential Equations*. Cambridge University Press, 1996.
- [7] Fuzhen Zhang, *The Schur Complement and its Applications*, Springer, Numerical Methods and Algorithms, Vol. 4, 2005.
- [8] B. I. Wohlmuth; *Discretization Methods and Iterative Solvers Based on Domain Decomposition*. Springer, 2003.
- [9] L. F. Pavarino, A. Toselli; *Recent Developments in Domain Decomposition Methods*. Springer, 2003.
- [10] M.B. Allen III, I. Herrera & G. F. Pinder; *Numerical Modeling in Science And Engineering*. John Wiley & Sons, Inc . 1988.
- [11] R. L. Burden y J. D. Faires; *Análisis Numérico*. Math Learning, 7 ed. 2004.
- [12] S. Friedberg, A. Insel, and L. Spence; *Linear Algebra*, 4th Edition, Prentice Hall, Inc. 2003.
- [13] Y. Saad; *Iterative Methods for Sparse Linear Systems*. SIAM, 2 ed. 2000.
- [14] Y. Skiba; *Métodos y Esquemas Numéricos, un Análisis Computacional*. UNAM, 2005.
- [15] W. Gropp, E. Lusk, A. Skjelle, *Using MPI, Portable Parallel Programming With the Message Passing Interface*. Scientific and Engineering Computation Series, 2ed, 1999.
- [16] I. Foster; *Designing and Building Parallel Programs*. Addison-Wesley Inc., Argonne National Laboratory, and the NSF, 2004.

- [17] Jorge L. Ortega-Arjona, *Patterns for Parallel Software Design*, Wiley series in Software Design Patterns, 2010.
- [18] DDM Organization, *Proceedings of International Conferences on Domain Decomposition Methods*, 1988-2012.
<http://www.ddm.org> and <http://www.domain-decomposition.com>
- [19] Toselli, A., and Widlund O. *Domain decomposition methods- Algorithms and theory*, Springer Series in Computational Mathematics, Springer-Verlag, Berlin, 2005, 450p.
- [20] Farhat, C. and Roux, F. X. *A Method of Finite Element Tearing and Interconnecting and its Parallel Solution Algorithm*. Int. J. Numer. Meth. Engrg., 32:1205-1227, 1991.
- [21] Mandel J. & Tezaur R. *Convergence of a Substructuring Method with Lagrange Multipliers*, Numer. Math. 73 (1996) 473-487.
- [22] Farhat C., Lesoinne M. Le Tallec P., Pierson K. & Rixen D. *FETI-DP a Dual-Primal Unified FETI method, Part 1: A Faster Alternative to the two-level FETI Method*, Int. J. Numer. Methods Engrg. 50 (2001) 1523-1544.
- [23] Farhat C., Lesoinne M., Pierson K. *A Scalable Dual-Primal Domain Decomposition Method*, Numer. Linear Algebra Appl. 7 (2000) 687-714.
- [24] Mandel J. & Tezaur R. *On the Convergence of a Dual-Primal Substructuring Method*, Numer. Math. 88(2001), pp. 5443-558.
- [25] Mandel, J. *Balancing Domain Decomposition*. Comm. Numer. Meth. Engrg., 9:233-241, 1993.
- [26] Mandel J., & Brezina M., *Balancing Domain Decomposition for Problems with Large Jumps in Coefficients*, Math. Comput. 65 (1996) 1387-1401.
- [27] Dohrmann C., *A Preconditioner for Substructuring Based on Constrained Energy Minimization*. SIAM J. Sci. Comput. 25 (2003) 246-258.
- [28] Mandel J. & Dohrmann C., *Convergence of a Balancing Domain Decomposition by Constraints and Energy Minimization*. Numer. Linear Algebra Appl. 10 (2003) 639-659.
- [29] Da Conceição, D. T. Jr., *Balancing Domain Decomposition Preconditioners for Non-symmetric Problems*, Instituto Nacional de Matemática pura e Aplicada, Agencia Nacional do Petróleo PRH-32, Rio de Janeiro, May. 9, 2006.
- [30] J. Li and O. Widlund, *FETI-DP, BDDC and block Cholesky Methods*, Int. J. Numer. Methods Engrg. 66, 250-271, 2005.

- [31] Farhat Ch., Lesoinne M., Le Tallec P., Pierson K. and Rixen D. *FETI-DP: A Dual-Primal Unified FETI Method-Part I: A Faster Alternative to the Two Level FETI Method*. Internal. J. Numer. Methods Engrg., 50:1523-1544, 2001.
- [32] Rixen, D. and Farhat Ch. *A Simple and Efficient Extension of a Class of Substructure Based Preconditioners to Heterogeneous Structural Mechanics Problems*. Internal. J. Numer. Methods Engrg., 44:489-516, 1999.
- [33] J. Mandel, C. R. Dohrmann, and R. Tezaur, *An Algebraic Theory for Primal and Dual Substructuring Methods by Constraints*, Appl. Numer. Math., 54 (2005), pp. 167-193.
- [34] A. Klawonn, O. B. Widlund, and M. Dryja, *Dual-primal FETI Methods for Three-Dimensional Elliptic Problems with Heterogeneous Coefficients*, SIAM J. Numer. Anal., 40 (2002), pp. 159-179.
- [35] Alberto Rosas Medina, *Métodos de Estabilización para Problemas de Advección-Difusión*, Trabajo de Investigación para Sustentar el Examen de Candidatura al Doctorado, Postgrado en Ciencias de la Tierra, UNAM, 2011.
- [36] Klawonn A. and Widlund O.B., *FETI and Neumann-Neumann Iterative Substructuring Methods: Connections and New Results*. Comm. Pure and Appl. Math. 54(1): 57-90, 2001.
- [37] Tezaur R., *Analysis of Lagrange Multipliers Based Domain Decomposition*. P.H. D. Thesis, University of Colorado, Denver, 1998.
- [38] Herrera I. & Rubio E., *Unified Theory of Differential Operators Acting on Discontinuous Functions and of Matrices Acting on Discontinuous Vectors*, 19th International Conference on Domain Decomposition Methods, Zhangjiajie, China 2009. (Oral presentation). Internal report #5, GMMC-UNAM, 2011.
- [39] Valeri I. Agoshkov, *Poincaré-Steklov Operators and Domain Decomposition Methods in Finite Dimensional Spaces*. First International Symposium on Domain Decomposition Methods for Partial Differential Equations, pages 73-112, Philadelphia, PA, 1988. SIAM. Paris, France, January 7-9, 1987.
- [40] Toselli, A., *FETI Domain Decomposition Methods for Escalar Advection-Diffusion Problems*. Computational Methods Appl. Mech. Engrg. 190. (2001), 5759-5776.
- [41] C.T. Keller, *Iterative Methods for Linear and Nonlinear Equations*, Societe for Industrial and Applied Mathematics, 1995.
- [42] Manoj Bhardwaj, David Day, Charbel Farhat, Michel Lesoinne, Kendall Pierson, and Daniel Rixen. *Application of the PETI Method to ASCI Problems: Scalability Results on One Thousand Processors and Discussion*

- of Highly Heterogeneous Problems*. Intemat. J. Numer. Methods Engrg., 47:513-535, 2000.
- [43] Zdeněk Dostál and David Hordk. *Scalability and FETI Based Algorithm for Large Discretized Variational Inequalities*. Math. Comput. Simulation, 61(3-6): 347-357, 2003. MODELLING 2001 (Pilsen).
- [44] Charbel Farhat, Michel Lesoinne, and Kendall Pierson. *A Scalable Dual-Primal Domain Decomposition Method*. Numer. Linear Algebra Appl., 7(7-8):687-714, 2000.
- [45] Yannis Fragakis and Manolis Papadrakakis, *The Mosaic of High Performance Domain Decomposition Methods for Structural Mechanics: Formulation, Interrelation and Numerical Efficiency of Primal and Dual Methods*. Comput. Methods Appl. Mech. Engrg, 192(35-36):3799-3830, 2003.
- [46] Kendall H. Pierson, *A family of Domain Decomposition Methods for the Massively Parallel Solution of Computational Mechanics Problems*. PhD thesis, University of Colorado at Boulder, Aerospace Engineering, 2000.
- [47] Manoj Bhardwaj, Kendall H. Pierson, Garth Reese, Tim Walsh, David Day, Ken Alvin, James Peery, Charbel Farhat, and Michel Lesoinne. Salinas, *A Scalable Software for High Performance Structural and Mechanics Simulation*. In ACM/IEEE Proceedings of SC02: High Performance Networking and Computing. Gordon Bell Award, pages 1-19, 2002.
- [48] Klawonn, A.; Rheinbach, O., *Highly Scalable Parallel Domain Decomposition Methods with an Application to Biomechanics*, Journal of Applied Mathematics and Mechanics 90 (1): 5-32, doi:10.1002/zamm.200900329.
- [49] Petter E. Bjørstad and Morten Skogen. *Domain Decomposition Algorithms of Schwarz Type, Designed for Massively Parallel Computers*. In David E. Keyes, Tony F. Chan, Gerard A. Meurant, Jeffrey S. Scroggs, and Robert G. Voigt, editors. Fifth International Symposium on Domain Decomposition Methods for Partial Differential Equations, pages 362-375, Philadelphia, PA, 1992. SIAM. Norfolk, Virginia, May 6-8, 1991.
- [50] Yau Shu Wong and Guangrui Li. *Exact Finite Difference Schemes for Solving Helmholtz Equation at any Wavenumber*. International Journal of Numerical Analysis and Modeling, Series B, Volume 2, Number 1, Pages 91-108, 2011.
- [51] Zhangxin Chen, Guanren Huan and Yuanle Ma. *Computational Methods for Multiphase Flow in Porous Media*, SIAM, 2006.