Introducción al Método de Diferencias Finitas y su Implementación Computacional

Antonio Carrillo Ledesma,
Karla Ivonne González Rosas y Omar Mendoza Bernal
Facultad de Ciencias, UNAM
http://www.mmc.geofisica.unam.mx/acl/

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

Primavera 2018, Versión 1.0β

Índice

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

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

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 la aproximación de la primera derivada

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

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

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

Es común escribir la expresión anterior como

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

o

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

para simplificar la notación.

1.1.2 Diferencias Regresivas

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

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

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

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

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

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

Es común escribir la expresión anterior como

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

o

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

para simplificar la 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

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

y

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

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

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

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

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

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

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

comparado el error local de truncamiento de la aproximación anterior $O_c(\Delta x^2)$, con los obtenidos previamente para diferencias progresivas $O_p(\Delta x)$ Ec.(1.5) y regresivas $O_r(\Delta x)$ Ec.(1.10), se tiene que

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

Es común encontrar expresada la derivada[1]

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

como

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

para simplificar la notación.

Derivadas Usando Más Puntos Utilizando el valor de la función en más puntos se construyen fórmulas más precisas para las derivadas[2], algunos ejemplos son

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

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.

[1] En 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. $\left.\frac{df}{dx}\right|_{x_i} = \frac{f(x_i+\Delta x_{i-}) - f(x_i - \Delta x_{i+})}{(\Delta x_{i-}) + (\Delta x_{i+})}$.

[2] Al 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

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

eliminando las primeras derivadas, 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[3]

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

Es común escribir la expresión anterior como

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

para simplificar la 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

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

[3] En 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. $f''(x_i) = \frac{f(x_i-\Delta x_{i-})-2f(x_i)+f(x_i+\Delta x_{i+})}{(\Delta x_{i-})(\Delta x_{i+})}$

1.2.2 Derivadas de Orden Tres y Cuatro

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

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

Algunos ejemplos para cuartas derivadas son

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

2 Método de Diferencias Finitas en Una Dimensión

Consideremos la ecuación diferencial parcial

$$(p(x)u'(x))' + q(x)u'(x) - r(x)u(x) = f(x) \tag{2.1}$$

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

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:

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 correspondientes 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 lineal algebraico de ecuaciones $\underline{\underline{A}}\underline{u} = \underline{f}$.
3. Resolver el sistema lineal algebraico de ecuaciones $\underline{\underline{A}}\underline{u} = \underline{f}$ (véase capítulo 5), 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

$$u''(x) = f(x), \qquad 0 \leq x \leq 1, \qquad u(0) = u_\alpha, \qquad u(1) = u_\beta \tag{2.2}$$

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

1. Generar una malla homogénea del dominio[4]

$$x_i = ih, \qquad i = 0, 1, ..., n, \qquad h = \frac{1}{n} = \Delta x \tag{2.3}$$

2. Sustituir la derivada con la 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[5]

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

[4] En 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. $u''(x_i) \approx \frac{u(x_i - h_{i-}) - 2u(x) + u(x_i + h_{i+})}{(h_{i-})(h_{i+})}$.

[5] Notemos 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}. El conjunto de estos tres puntos de la malla es comúnmente llamado el esténcil de diferencias finitas en una dimensión.

o en su forma simplificada

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

definiendo la solución aproximada de $u(x)$ en x_i como u_i, entonces se genera el siguiente sistema lineal de ecuaciones

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

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

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

en este caso, podemos factorizar $1/h^2$ del sistema lineal $\underline{\underline{A}}\underline{u} = \underline{f}$, quedando como

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

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 5.2) principalmente cuando $h \to 0$.

3. Resolver el sistema lineal algebraico de ecuaciones $\underline{\underline{A}}\underline{u} = \underline{f}$ (véase capítulo 5), obtenemos la solución aproximada en cada punto interior de la malla. La solución completa al problema la obtenemos al formar el vector

$$\left[\begin{array}{ccccccccc} u_\alpha & u_1 & u_2 & u_3 & \cdots & u_{n-2} & u_{n-1} & u_\beta \end{array} \right].$$

Uno de los paquetes más conocidos y usados para el cálculo numérico es MatLab[6] el cual tiene un alto costo monetario para los usuarios. Por ello, una opción es usar alternativas desarrolladas usando licencia de código abierto, un par de estos paquetes son: GNU OCTAVE[7] y SCILAB[8].

Octave corre gran parte del código desarrollado para MatLab sin requerir cambio alguno, en cuanto a SCILAB es requerido hacer algunos ajustes en la codificación y ejecución. En los siguientes ejemplos[9] se mostrará como implementar la solución computacional. En la sección (6) se mostrará la implementación en diversos paquetes y lenguajes de programación.

Ejemplo 1 *Sea*

$$-u''(x) = -\pi^2 \cos(\pi x), \qquad xi \leq x \leq xf, \qquad u(xi) = vi, \qquad u(xf) = vf$$

El programa queda implementado en OCTAVE (MatLab) como:

```
function [A,b,x] = fdm1d(xi,xf,vi,vf,n)
  N=n-2; % Nodos interiores
  h=(xf-xi)/(n-1); % Incremento en la malla
  %A = sparse(N,N); % Matriz SPARSE
  A=zeros(N,N); % Matriz A
  b=zeros(N,1); % 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;
```

[6] MatLab es un programa comercial para el cálculo numérico el cual provee un poderoso ambiente de cálculo para aplicaciones Científicas y de Ingeniería [http://www.mathworks.com.products/matlab.html].

[7] GNU OCTAVE 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.gnu.org/software/octave].

[8] SCILAB 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].

[9] Los ejemplos que se muestran en el presente texto se pueden descargar de la página WEB:

http://mmc.geofisica.unam.mx/acl/MDF/

o desde GitHub mediante

git clone git://github.com/antoniocarrillo69/MDF.git

```
  A(1,2)=Q;
  b(1)=LadoDerecho(xi)-vi*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(xi+h*(i-1));
  end
  % Renglon final de la matriz A y vector b
  A(N,N-1)=R;
  A(N,N)=P;
  b(N)=LadoDerecho(xi+h*N)-vf*Q;
  % Resuleve el sistema lineal Ax=b
  x=inv(A)*b;
  % Prepara la graficacion
  xx=zeros(n,1);
  zz=zeros(n,1);
  for i=1:n
    xx(i)=xi+h*(i-1);
    zz(i)=SolucionAnalitica(xx(i));
  end
  yy=zeros(n,1);
  yy(1)=vi; % Condicion inicial
  for i=1:N
    yy(i+1)=x(i);
  end
  yy(n)=vf; % Condicion inicial
  % Grafica la solucion de la Ecuacion Diferencial Parcial en 1D
  plot(xx,[yy,zz]);
endfunction
function y=LadoDerecho(x)
  y=-pi*pi*cos(pi*x);
endfunction
function y=SolucionAnalitica(x)
  y=cos(pi*x);
endfunction
```

Si el programa lo grabamos en el directorio de trabajo con el nombre **fdm1d.m**, entonces se puede ejecutar en la consola de OCTAVE (MatLab) mediante

$$[A, b, x] = fdm1d(-1, 2, -1, 1, 30);$$

donde es necesario indicar el inicio (-1) y fin (2) del dominio, el valor de la condición de frontera de inicio (-1) y fin (1), además de el número de nodos (30) en la partición. La ejecución genera la gráfica de la solución calculada por

el método de diferencias finitas y la solución analítica en los mismos puntos de la malla, además devuelve la matriz[10] y los vectores A, b, x generados por la función.

En OCTAVE (MatLab) es posible introducir funciones para que pasen cómo parámetros a otra función, de tal forma que se puede definir un programa genérico de diferencias finitas en una dimensión con condiciones de frontera Dirichlet en el cual se especifiquen los parámetros de la ecuación en la línea de comando de OCTAVE.

Usando este hecho, implementamos un programa para codificar el Método de Diferencias Finitas en una Dimensión para resolver el problema con condiciones Dirichlet

$$p(x)u'' + q(x)u' + r(x)u = f(x)$$

definido en el dominio $[xi, xf]$ y valores en la frontera de $u(xi) = vi$ y $u(xf) = vf$, además se le indica el tamaño de la partición n, si se graficará la solución indicando en $grf = 1$, con solución analítica $s(x)$ y si a ésta se proporciona entonces $sws = 1$. Regresando la matriz y los vectores $\underline{\underline{A}}\underline{u} = \underline{b}$ del sistema lineal generado así como los puntos del dominio y los valores de la solución en dichos puntos x, V, para cada problema que deseemos resolver.

Ejemplo 2 *El programa queda implementado en OCTAVE (MatLab) como:*

```
function [error,A,b,u,x,V] = fdm1d_DD(p,q,r,f,xi,xf,vi,vf,n,grf,s,sws)
    if n < 3
        return
    end
    % Numero de incognitas del problema
    tm = n -2;
    % Declaracion de la matriz y vectores de trabajo
    %A = sparse(tm,tm);
    A = zeros(tm,tm); % Matriz de carga
    b = zeros(tm,1); % Vector de carga
    u = zeros(tm,1); % Vector de solucion
    x = zeros(n,1); % Vector de coordenadas de la particion
    V = zeros(n,1) ; % Vector solucion
    h = (xf-xi)/(n-1);
    h1 = h*h;
    % Llenado de los puntos de la malla
    for i = 1: n,
        x(i) = xi + (i-1)*h;
    end
```

[10] En Octave (MatLab) se define a una matriz mediante A = zeros(N,N), este tipo de matriz no es adecuada para nuestros fines (véase capítulo 5). Para ahorrar espacio y acelerar los cálculos numéricos que se requieren para resolver el sistema lineal asociado usamos un tipo de matriz que no guarda valores innecesarios (ceros), esto se hace mediante la declaración de la matriz como A = sparse(N,N).

```
% Llenado de la matriz y vector
b(1) = f(xi) - p(xi)*(vi/h1);
A(1,2) = p(x(1))/h1 - q(x(1))/(2.0*h);
A(1,1) = ((-2.0 * p(x(1))) / h1) + r(x(1));
for i=2:tm-1,
    A(i,i-1) = p(x(i))/h1 - q(x(i))/(2.0*h);
    A(i,i) = ((-2.0 * p(x(i))) / h1) + r(x(i));
    A(i,i+1) = p(x(i))/h1 + q(x(i))/(2.0*h);
    b(i) = f(x(i));
end
A(tm,tm-1) = p(x(tm))/h1 - q(x(tm))/(2.0*h);
A(tm,tm) = ((-2.0 * p(x(tm))) / h1) + r(x(tm));
b(tm) = f(x(tm+1))-p(x(tm+1))*(vf/h1);
% Soluciona el sistema
u = inv(A)*b;
% Copia la solucion obtenida del sistema lineal al vector solucion
V(1) = vi;
for i=1:tm,
    V(i+1) = u(i);
end
V(n) = vf;
% Encuentra el error en norma infinita usando una particion par = 10
error = 0;
if sws == 1
    par = 10;
    for i = 1: n-1,
        inc = (x(i+1)-x(i))/par;
        for j = 1:par+1,
            px = x(i)+inc*(j-1);
            e = abs(s(px)-l(px,x(i),V(i),x(i+1),V(i+1)));
            if e > error
                error = e;
            end
        end
    end
end
% Revisa si se graficara
if grf == 1
    if sws == 1
        % Calcula la solucion analitica en la particion de calculo
        ua = zeros(n,1);
        for i = 1: n,
            ua(i) = s(x(i));
        end
    end
    % Graficar la solucion numerica
```

```
            plot(x,V,'o');
            hold
            % Graficar la solucion analitica en una particion de tamano xPart
            if sws == 1
                xPart = 1000;
                h = (xf-xi)/(xPart-1);
                xx = zeros(xPart,1);
                xa = zeros(xPart,1);
                for i = 1: xPart,
                    xx(i) = xi + (i-1)*h;
                    xa(i) = s(xx(i));
                end
                plot(xx,xa);
                % Grafica el error
                figure(2);
                plot(x,V-ua);
            end
        end
    end
    % Evalua el punto x en la recta dada por los puntos (x1,y1) y (x2,y2), se
usa para el calculo de la norma infinito
    function y = l(x,x1,y1,x2,y2)
        y = y1+((y2-y1)/(x2-x1))*(x-x1);
    end
```

De esta forma podemos implementar diversos problemas y ver su solución

Ejemplo 3 *Sea*

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

para ejecutar el programa anterior es necesario escribir en la consola de OCTAVE:

```
p=@(x) 1;
q=@(x) 0;
r=@(x) 0;
f=@(x) -pi*pi*cos(pi*x);
s=@(x) cos(pi*x);
[error,A,b,u,x,V] = fdm1d_ DD(p,q,r,f,-1,2,-1,1,11,1,s,1);
```

Otro ejemplo:

Ejemplo 4 *Sea*

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

para ejecutar el programa anterior es necesario escribir en la consola de OCTAVE:

```
p=@(x) 1;
q=@(x) 0;
r=@(x) 0;
f=@(x) -pi*pi*cos(pi*x);
s=@(x) cos(pi*x);
[error,A,b,u,x,V] = fdm1d_DD(p,q,r,f,0,1,1,-1,40,1,s,1);
```

Ahora uno con un parámetro adicional (velocidad):

Ejemplo 5 *Sea*

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

para ejecutar el programa anterior es necesario escribir en la consola de OCTAVE:

```
v=@(x) 1.0; % velocidad
p=@(x) -1.0;
q=@(x) v(x);
r=@(x) 0;
f=@(x) 0;
s=@(x) (exp(v(x)*x)-1.0)/(exp(v(x))-1.0);
[error,A,b,u,x,V] = fdm1d_DD(p,q,r,f,0,1,0,1,11,1,s,1);
```

En éste caso, la velocidad es $v = 1.0$ y nuestro programa lo puede resolver sin complicación alguna. Si aumentamos la velocidad a $v = 50.0$ y volvemos a correr el programa haciendo estos cambios, entonces:

Ejemplo 6

```
v=@(x) 50.0;
[error,A,b,u,x,V] = fdm1d_DN(p,q,r,f,0,1,-1,0,-1,1,11,1,s,1);
```

Aún el programa lo resuelve bien, pero si ahora subimos la velocidad a $v = 100.0$ y corremos nuevamente el programa, entonces:

Ejemplo 7

```
v=@(x) 100.0;
[error,A,b,u,x,V] = fdm1d_DN(p,q,r,f,0,1,-1,0,-1,1,11,1,s,1);
```

Entonces tendremos que la solución oscila en torno al cero. Para corregir esto, tenemos algunas opciones: Aumentar el número de nodos en la malla de discretización, usar una malla no homogénea refinada adecuadamente para tener un mayor número de nodos en donde sea requerido, usar fórmulas más precisas para las derivadas o mejor aún, usar diferentes esquemas tipo Upwind, Scharfetter-Gummel y Difusión Artificial para implementar el Método de Diferencias Finitas, como se muestra a continuación.

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

$$\left(p\left(x\right)u'\left(x\right)\right)' + q\left(x\right)u'\left(x\right) - r\left(x\right)u\left(x\right) = f\left(x\right) \tag{2.7}$$

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

el término $q\left(x\right)u'\left(x\right)$ algunas veces es llamado el término advectivo[11] si u es la velocidad y puede ocasionar inestabilidad numérica en el Método de Diferencias Finitas como se mostró en el ejemplo anterior Ejemp.(7), para evitar dichas inestabilidades existen diversas técnicas de discretización mediante el análisis de la solución considerando el Número de Péclet (local y global) y su implementación en el Método de Diferencias Finitas (véase [35]) mediante el esquema Upwind, Scharfetter-Gummel y Difusión Artificial, entre otros, para ello consideremos:

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

$$p_i\frac{u_{i-1} - 2u_i + u_{i+1}}{h^2} + q_i\frac{u_{i+1} - u_{i-1}}{2h} - r_iu_i = f_i \qquad i = 1, 2, ..., n. \tag{2.8}$$

  donde la ventaja de ésta 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 $r\left(x\right) > 0$ y $p\left(x\right) > 0$. Cuando la advección $\left|p\left(x\right)\right|$ 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, ésto se logra mediante la siguiente discretización para $\left(p\left(x,t\right)u'\left(x,t\right)\right)'$ mediante

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

  entonces se tiene que

$$\frac{p(u_{i+1},t)\frac{u_{i+1}+u_i}{h} - p\left(u_{i-1,t}\right)\frac{u_i - u_{i-1}}{h}}{h} + q_i\frac{u_{i+1} - u_{i-1}}{2h} - r_iu_i = f_i \tag{2.10}$$

  para $i = 1, 2, ..., n$, (véase [52] pág. 78 y 79).

[11] Cuando la advección es fuerte, esto es cuando $\left|q\left(x\right)\right|$ 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

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

  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.

2.2 Problema con Condiciones de Frontera Neumann

Consideremos el problema

$$u''(x) = f(x), \qquad 0 \leq x \leq 1 \tag{2.12}$$

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, primero debemos discretizar las condiciones de frontera, una manera sería 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

$$\frac{u_1-u_0}{h} = cte_1 \tag{2.13}$$

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

$$\frac{u_n-u_{n-1}}{h} = cte_2 \tag{2.14}$$

pero el orden de aproximación no sería 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

$$\frac{u_1 - u_{-1}}{2h} = cte_1 \tag{2.15}$$

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

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

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

1. Generar una malla homogénea del dominio

$$x_i = ih, \qquad i = 0, 1, ..., n, \qquad h = \frac{1}{n} = \Delta x. \tag{2.17}$$

2. Sustituir la derivada con la Ec.(1.24)[12] 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

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

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

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

[12] Para mantener la estabilidad es necesario tomar en cuenta las distintas formas de discretización en el Método de Diferencias Finitas mediante el esquema Upwind, Scharfetter-Gummel y Difusión Artificial, entre otros (véase sección: 2.1, Ecs. 2.8 a 2.11).

3. Resolver el sistema lineal algebraico de ecuaciones $\underline{\underline{A}}\underline{u} = \underline{f}$ (véase capítulo 5), 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 mixta, también conocidas como condiciones de frontera Robin. Sin perdida 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

$$u''(x) = f(x), \qquad 0 \leq x \leq 1 \tag{2.20}$$

con condiciones de frontera Dirichlet y Neumann

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

respectivamente. Para usar el procedimiento general de solución numérica mediante el método de diferencias finitas, primero debemos 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

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

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

Entonces debemos de:

1. Generar una malla homogénea del dominio

$$x_i = ih, \qquad i = 0, 1, ..., n, \qquad h = \frac{1}{n} = \Delta x. \tag{2.22}$$

2. Sustituir la derivada con la Ec.(1.24)[13] 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

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

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. Resolver el sistema lineal algebraico de ecuaciones $\underline{\underline{A}}\underline{u} = \underline{f}$ (véase capítulo 5), obtenemos la solución aproximada en cada punto de la malla. La solución completa al problema la obtenemos al formar el vector

$$\left[\begin{array}{ccccccccc} u_\alpha & u_1 & u_2 & u_3 & \cdots & u_{n-2} & u_{n-1} & u_n \end{array} \right].$$

La implementación de un programa para codificar el Método de diferencias Finitas en una Dimensión para resolver el problema con condiciones Dirichlet o Neumann del problema

$$p(x)u'' + q(x)u' + r(x)u = f(x)$$

definido en el dominio $[xi, xf]$ y el tipo de frontera en xi —ti igual a -1 Dirichlet ($u(xi) = vi$) ó -2 Neumann ($u_x(xi) = vi$)— y el valor en la frontera xf —tf

[13] Para mantener la estabilidad es necesario tomar en cuenta las distintas formas de discretización en el Método de Diferencias Finitas mediante el esquema Upwind, Scharfetter-Gummel y Difusión Artificial, entre otros (véase sección: 2.1, Ecs. 2.8 a 2.11).

igual a -1 Dirichlet($u(xf) = vf$) o -2 Neumann ($u_x(xf) = vf$)—, además se le indica el tamaño de la partición n, si se graficará la solución indicando en $grf = 1$, con la solución analítica $s(x)$ y si ésta se proporciona $sws = 1$. Regresando la matriz y los vectores $\underline{\underline{A}}\underline{u} = \underline{b}$ del sistema lineal generado así como los puntos del dominio y los valores de la solución en dichos puntos x, V

Ejemplo 8 *El programa queda implementado en OCTAVE (MatLab) como:*

```
function [error,A,b,u,x,V] = fdm1d(p,q,r,f,xi,xf,ti,vi,tf,vf,n,grf,s,sws)
    if n < 3
        return
    end
    % llenado de valores para el tipo de nodo y valor de frontera
    TN = zeros(n,1); % Vector para guardar el tipo de nodo
    V = zeros(n,1); % Vector para guardar los valores de la solucion y
frontera
    TN(1) = ti;
    TN(n) = tf;
    V(1) = vi;
    V(n) = vf;
    % Calcula el numero de incognitas del problema
    tm = 0;
    for i=1:n,
        if TN(i) == -1 % Descarta nodos frontera Dirichlet
            continue
        end
        tm = tm +1;
    end
    % Declaracion de la matriz y vectores de trabajo
    %A = sparse(tm,tm);
    A = zeros(tm,tm); % Matriz de carga
    b = zeros(tm,1); % Vector de carga
    u = zeros(tm,1); % Vector de solucion
    x = zeros(n,1); % Vector de coordenadas de la particion
    % Llenado de la matriz y vector
    h = (xf-xi)/(n-1);
    h1 = h*h;
    j = 1;
    for i=1:n,
        x(i) = xi + (i-1)*h;
        if TN(i) == -1 % Descarta nodos frontera Dirichlet
            continue;
        end
        % Diagonal central
        A(j,j) = ((-2.0 * p(x(i))) / h1) + r(x(i));
        if TN(i) == -2
            A(j,j) = -1/h1;
```

```
        end
        % Lado derecho
        b(j) = f(x(i));
        % Diagonal anterior
        if TN(i) == -2
            b(j) = V(i)/h;
            if i > 1
                A(j,j-1) = -1/h1;
            end
        elseif TN(i-1) == -1
            b(j) = f(x(i)) - p(x(i))*(V(i-1)/h1);
        else
            A(j,j-1) = p(x(i))/h1 - q(x(i))/(2.0*h);
        end
        % Diagonal superior
        if TN(i) == -2
            b(j) = V(i)/h;
            if i < n
                A(j,j+1) = -1/h1;
            end
        elseif TN(i+1) == -1
            b(j) = f(x(i)) - p(x(i))*(V(i+1)/h1);
        else
            A(j,j+1) = p(x(i))/h1 + q(x(i))/(2.0*h);
        end
        j = j + 1;
    end
    % Soluciona el sistema
    u = A\b;
    % Copia la solucion obtenida del sistema lineal al vector solucion
    j = 1;
    for i=1:n,
        if TN(i) == -1 % descarta nodos frontera Dirichlet
            continue
        end
        V(i) = u(j);
        j = j + 1;
    end
    % Encuentra el error en norma infinita usando una particion par = 10
    error = 0;
    if sws == 1
        par = 10;
        for i = 1: n-1,
            inc = (x(i+1)-x(i))/par;
            for j = 1:par+1,
                px = x(i)+inc*(j-1);
```

```
                    e = abs(s(px)-l(px,x(i),V(i),x(i+1),V(i+1)));
                    if e > error
                        error = e;
                    end
                end
            end
        end
        % Revisa si se graficara
        if grf == 1
            if sws == 1
                % Calcula la solucion analitica en la particion de calculo
                ua = zeros(n,1);
                for i = 1: n,
                    ua(i) = s(x(i));
                end
            end
            % Graficar la solucion numerica
            plot(x,V,'o');
            hold
            % Graficar la solucion analitica en una particion de tamano xPart
            if sws == 1
                xPart = 1000;
                h = (xf-xi)/(xPart-1);
                xx = zeros(xPart,1);
                xa = zeros(xPart,1);
                for i = 1: xPart,
                    xx(i) = xi + (i-1)*h;
                    xa(i) = s(xx(i));
                end
                plot(xx,xa);
                % Grafica el error
                figure(2);
                plot(x,V-ua);
            end
        end
    end
    % evalua el punto x en la recta dada por los puntos (x1,y1) y (x2,y2)
    function y = l(x,x1,y1,x2,y2)
        y = y1+((y2-y1)/(x2-x1))*(x-x1);
    end
```

Ejemplo 9 *Sea*

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

para ejecutar el programa anterior es necesario escribir en la consola de OCTAVE:

```
v=@(x) 1.0; % velocidad, posibles valores 1,25,100, etc
p=@(x) -1.0;
q=@(x) v(x);
r=@(x) 0;
f=@(x) 0;
s=@(x) (exp(v(x)*x)-1.0)/(exp(v(x))-1.0);
[error,A,b,u,x,V] = fdm1d_DN(p,q,r,f,0,1,-1,0,-1,1,11,1,s,1);
```

Ejemplo 10 *Sea*

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

para ejecutar el programa anterior es necesario escribir en la consola de OCTAVE:

```
p=@(x) 1;
q=@(x) 0;
r=@(x) 0;
f=@(x) -pi*pi*cos(pi*x);
s=@(x) cos(pi*x);
[error,A,b,u,x,V] = fdm1d_DN(p,q,r,f,0,1,-1,1,-1,-1,40,1,s,1);
```

Ejemplo 11 *Sea*

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

para ejecutar el programa anterior es necesario escribir en la consola de OCTAVE:

```
p=@(x) 1;
q=@(x) 0;
r=@(x) 0;
f=@(x) -pi*pi*cos(pi*x);
s=@(x) cos(pi*x);
[error,A,b,u,x,V] = fdm1d_DN(p,q,r,f,0,0.5,-1,1,-2,-pi,40,1,s,1);
```

Pese a que con el programa anterior podríamos resolver la ecuación

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

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 correctamente dicha ecuación, usaremos otro programa con los esquemas de discretización de diferencias finitas y diferencias finitas exactas. Este último procedimiento fue 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. Y la codificación de prueba queda como:

Ejemplo 12 *Sea*

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

con $k = 150$, *entonces el programa en SCILAB queda implementado como:*

```
TEST = 1; // (0) Diferencias finitas, (1) Diferencias finitas exactas

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; // Particion
N=M-1; // Nodos interiores
h=(c-a)/(M-1); // Incremento en la malla
Y0=1; // Condicion Dirchlet inicial en el inicio del dominio
Y1=%i*K; // Condicion 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

// 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
// Renglon 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

// Resuleve 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 graficacion
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
// Grafica la solucion 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

$$u_t = \mathcal{L}u \tag{2.25}$$

donde

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

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

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

Existen diferentes casos del esquema $theta$ a saber:

- Para $\theta = 0$, se obtiene un esquema de diferencias finitas hacia adelante 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[14]:

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

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

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

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:

$$Au^{j+1} = Bu^j + f^j \tag{2.31}$$

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 SCILAB[15]

Ejemplo 13 *Sea*

$$u_t - a(x)u''(x) - b(x)u'(x) + c(x)u = f, \qquad l0 \leq x \leq l, \qquad 0 < t < T$$

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
// intrevalo [l0, l]
// entero m>=3
// entero N>=1
```

[14] Para mantener la estabilidad es necesario tomar en cuenta las distintas formas de discretización en el Método de Diferencias Finitas mediante el esquema Upwind, Scharfetter-Gummel y Difusión Artificial, entre otros (véase sección: 2.1, Ecs. 2.8 a 2.11).

[15] Scilab 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].

```
// Salida
// aproximaciones w_ij a u(x_i,t_j)
// Funciones de los coeficientes
function y = a(x)
y = -1;
endfunction
function y = b(x)
y = -1;
endfunction
function y = c(x)
y = 1;
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[16]

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

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

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

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

$$u_i^{j+1} = 2u_i^j - u_i^{j-1} + \left(\Delta t\right)^2 Bu^j \tag{2.35}$$

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

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

para $j = 0$, entonces

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

donde 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, para el primer tiempo tenemos

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

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

[16] Para mantener la estabilidad es necesario tomar en cuenta las distintas formas de discretización en el Método de Diferencias Finitas mediante el esquema Upwind, Scharfetter-Gummel y Difusión Artificial, entre otros (véase sección: 2.1, Ecs. 2.8 a 2.11).

La derivación del método parte de

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

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 14 *Sea*

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

sujeta a

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

con solución analítica

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

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 guradaria 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 $U = [U_1, U_2, ..., U_n]^T$ el vector solución obtenido al utilizar el método de diferencias finitas y $u = [u(x_1), u(x_n), ..., u(x_n)]$ 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 1 *Si* $\|E\| \leq Ch^p, p > 0$, *decimos que el método de diferencias finitas es de orden-p de precisión.*

Definición 2 *Un método de diferencias finitas es llamado convergente si*

$$\lim_{h \to 0} \|E\| = 0. \tag{2.40}$$

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 $u''(x) = f(x)$, uno de los operadores de diferencias finitas puede ser

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

Definición 3 *El error de truncamiento local es definido como*

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

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

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

Note que el error de truncamiento local sólo depende de la solución del esténcil 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 4 *Un esquema de diferencias finitas es llamado consistente si*

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

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

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

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 que 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

$$Au = F + T, \qquad AU = F, \qquad A(u - U) = T = -AE \tag{2.46}$$

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 $\|E\| = \left\|A^{-1}T\right\| \leq \left\|A^{-1}\right\| \|T\|$. Para el esquema de diferencias finitas centradas, tenemos que $\|E\| = \left\|A^{-1}\right\| h^2$. Tal que el error global depende del error de truncamiento local y $\left\|A^{-1}\right\|$.

Definición 5 *Un método de diferencias finitas para la ecuación diferencial elíptica es estable si A es invertible y*

$$\left\|A^{-1}\right\| \leq C, \text{ para todo } h \leq h_0 \tag{2.47}$$

donde C y h_0 son constantes.

Teorema 6 *Si el método de diferencias finitas es estable y consistente, entonces es convergente.*

3 Método de Diferencias Finitas en Dos y Tres Dimensiones

Consideremos la ecuación diferencial parcial

$$\left(p\left(x\right)u'\left(x\right)\right)' + q\left(x\right)u'\left(x\right) - r\left(x\right)u\left(x\right) = f\left(x\right) \tag{3.1}$$

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

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:

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 correspondientes 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 $\underline{\underline{A}}\underline{u} = \underline{f}$.
3. Resolver el sistema de ecuaciones (véase capítulo 5), y así obtener la solución aproximada en cada punto de la malla.

3.1 Derivadas en Dos y Tres Dimensiones

De forma análoga se construyen aproximaciones en diferencias finitas de primer y segundo orden en dos y tres dimensiones.

3.1.1 Derivadas en Dos Dimensiones

Usando el teorema de Taylor para funciones en dos variables x y y, es posible escribir de forma exacta para el punto x_i y y_j

$$\begin{aligned} f(x_i + \Delta x, y_j) &= f(x_i, y_j) + \Delta x \frac{\partial f\left(x_i, y_j\right)}{\partial x} + \frac{\Delta x}{2}\frac{\partial^2 f\left(x_i + \theta_1 \Delta x, y_j\right)}{\partial x^2} \\ f(x_i, y_j + \Delta y) &= f(x_i, y_j) + \Delta y \frac{\partial f\left(x_i, y_j\right)}{\partial y} + \frac{\Delta y}{2}\frac{\partial^2 f\left(x_i, y_j + \theta_2 \Delta y\right)}{\partial y^2}. \end{aligned} \tag{3.2}$$

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

$$\begin{aligned} \frac{\partial f\left(x_i, y_j\right)}{\partial x} &\simeq \frac{f\left(x_i + \Delta x, y_j\right) - f\left(x_i, y_j\right)}{\Delta x} \\ \frac{\partial f\left(x_i, y_j\right)}{\partial y} &\simeq \frac{f\left(x_i, y_j + \Delta y\right) - f\left(x_i, y_j\right)}{\Delta y} \end{aligned} \tag{3.3}$$

o en su forma simplificada (asociamos $\Delta x = h$ y $\Delta y = k$), entonces tenemos

$$\begin{aligned} \frac{\partial f(x_i, y_j)}{\partial x} &\simeq \frac{f_{i+1,j} - f_{i,j}}{h} \\ \frac{\partial f(x_i, y_j)}{\partial y} &\simeq \frac{f_{i,j+1} - f_{i,j}}{k}. \end{aligned} \tag{3.4}$$

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

$$\begin{aligned} \frac{\partial f(x_i, y_j)}{\partial x} &\simeq \frac{f(x_i, y_j) - f(x_i - \Delta x, y_j)}{\Delta x} \\ \frac{\partial f(x_i, y_j)}{\partial y} &\simeq \frac{f(x_i, y_j) - f(x_i, y_j - \Delta y)}{\Delta y} \end{aligned} \tag{3.5}$$

o en su forma simplificada tenemos

$$\begin{aligned} \frac{\partial f(x_i, y_j)}{\partial x} &\simeq \frac{f_{i,j} - f_{i-1,j}}{h} \\ \frac{\partial f(x_i, y_j)}{\partial y} &\simeq \frac{f_{i,j} - f_{i,j-1}}{k}. \end{aligned} \tag{3.6}$$

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

$$\begin{aligned} \frac{\partial f(x_i, y_j)}{\partial x} &\simeq \frac{f(x_i + \Delta x, y_j) - f(x_i - \Delta x, y_j)}{2\Delta x} \\ \frac{\partial f(x_i, y_j)}{\partial y} &\simeq \frac{f(x_i, y_j + \Delta y) - f(x_i, y_j - \Delta y)}{2\Delta y} \end{aligned} \tag{3.7}$$

o en su forma simplificada tenemos

$$\begin{aligned} \frac{\partial f(x_i, y_j)}{\partial x} &\simeq \frac{f_{i+1,j} - f_{i-1,j}}{2h} \\ \frac{\partial f(x_i, y_j)}{\partial y} &\simeq \frac{f_{i,j+1} - f_{i,j-1}}{2k}. \end{aligned} \tag{3.8}$$

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

$$\begin{aligned} \frac{\partial^2 f(x_i, y_j)}{\partial x^2} &\simeq \frac{f(x_i - \Delta x, y_j) - 2f(x_i, y_j) + f(x_i + \Delta x, y_j)}{\Delta x^2} \\ \frac{\partial^2 f(x_i, y_j)}{\partial y^2} &\simeq \frac{f(x_i, y_j - \Delta y) - f(x_i, y_j) + f(x_i, y_j + \Delta y)}{\Delta y^2} \end{aligned} \tag{3.9}$$

o en su forma simplificada tenemos

$$\begin{aligned} \frac{\partial^2 f(x_i, y_j)}{\partial x^2} &\simeq \frac{f_{i-1,j} - 2f_{i,j} + f_{i+1,j}}{h^2} \\ \frac{\partial^2 f(x_i, y_j)}{\partial y^2} &\simeq \frac{f_{i,j-1} - 2f_{i,j} + f_{i,j+1}}{k^2}. \end{aligned} \tag{3.10}$$

3.1.2 Derivadas en Tres Dimensiones

Usando el teorema de Taylor 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

$$\begin{aligned} f(x_i+\Delta x, y_j, z_k) &= f(x_i, y_j, z_k) + \Delta x \frac{\partial f(x_i, y_j, z_k)}{\partial x} + \frac{\Delta x}{2} \frac{\partial^2 f(x_i+\theta_1 \Delta x, y_j, z_k)}{\partial x^2} \\ f(x_i, y_j+\Delta y, z_k) &= f(x_i, y_j, z_k) + \Delta y \frac{\partial f(x_i, y_j, z_k)}{\partial y} + \frac{\Delta y}{2} \frac{\partial^2 f(x_i, y_j+\theta_2 \Delta y, z_k)}{\partial y^2} \\ f(x_i, y_j, z_k+\Delta z) &= f(x_i, y_j, z_k) + \Delta z \frac{\partial f(x_i, y_j, z_k)}{\partial z} + \frac{\Delta z}{2} \frac{\partial^2 f(x_i, y_j, z_k+\theta_3 \Delta z)}{\partial z^2} \end{aligned} \tag{3.11}$$

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

$$\begin{aligned} \frac{\partial f(x_i, y_j, z_k)}{\partial x} &\simeq \frac{f(x_i+\Delta x, y_j, z_k) - f(x_i, y_j, z_k)}{\Delta x} \\ \frac{\partial f(x_i, y_j, z_k)}{\partial y} &\simeq \frac{f(x_i, y_j+\Delta y, z_k) - f(x_i, y_j, z_k)}{\Delta y} \\ \frac{\partial f(x_i, y_j, z_k)}{\partial z} &\simeq \frac{f(x_i, y_j, z_k+\Delta z) - f(x_i, y_j, z_k)}{\Delta z} \end{aligned} \tag{3.12}$$

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

$$\begin{aligned} \frac{\partial f(x_i, y_j, z_k)}{\partial x} &\simeq \frac{f_{i+1,j,k} - f_{i,j,k}}{h} \\ \frac{\partial f(x_i, y_j, z_k)}{\partial y} &\simeq \frac{f_{i,j+1,k} - f_{i,j,k}}{l} \\ \frac{\partial f(x_i, y_j, z_k)}{\partial z} &\simeq \frac{f_{i,j,k+1} - f_{i,j,k}}{m}. \end{aligned} \tag{3.13}$$

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

$$\begin{aligned} \frac{\partial f(x_i, y_j, z_k)}{\partial x} &\simeq \frac{f(x_i, y_j, z_k) - f(x_i-\Delta x, y_j, z_k)}{\Delta x} \\ \frac{\partial f(x_i, y_j, z_k)}{\partial y} &\simeq \frac{f(x_i, y_j, z_k) - f(x_i, y_j-\Delta y, z_k)}{\Delta y} \\ \frac{\partial f(x_i, y_j, z_k)}{\partial z} &\simeq \frac{f(x_i, y_j, z_k) - f(x_i, y_j, z_k-\Delta z)}{\Delta z} \end{aligned} \tag{3.14}$$

o en su forma simplificada tenemos

$$\begin{aligned} \frac{\partial f(x_i, y_j, z_k)}{\partial x} &\simeq \frac{f_{i,j,k} - f_{i-1,j,k}}{h} \\ \frac{\partial f(x_i, y_j, z_k)}{\partial y} &\simeq \frac{f_{i,j,k} - f_{i,j-1,k}}{l} \\ \frac{\partial f(x_i, y_j, z_k)}{\partial z} &\simeq \frac{f_{i,j,k} - f_{i,j,k-1}}{m}. \end{aligned} \tag{3.15}$$

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

$$\begin{aligned}
\frac{\partial f\left(x_i, y_j, z_k\right)}{\partial x} &\simeq \frac{f\left(x_i+\Delta x, y_j, z_k\right)-f\left(x_i-\Delta x, y_j, z_k\right)}{2\Delta x} \\
\frac{\partial f\left(x_i, y_j, z_k\right)}{\partial y} &\simeq \frac{f\left(x_i, y_j+\Delta y, z_k\right)-f\left(x_i, y_j-\Delta y, z_k\right)}{2\Delta y} \\
\frac{\partial f\left(x_i, y_j, z_k\right)}{\partial z} &\simeq \frac{f\left(x_i, y_j, z_k+\Delta z\right)-f\left(x_i, y_j, z_k-\Delta z\right)}{2\Delta z}
\end{aligned} \tag{3.16}$$

o en su forma simplificada tenemos

$$\begin{aligned}
\frac{\partial f\left(x_i, y_j, z_k\right)}{\partial x} &\simeq \frac{f_{i+1,j,k}-f_{i-1,j,k}}{2h} \\
\frac{\partial f\left(x_i, y_j, z_k\right)}{\partial y} &\simeq \frac{f_{i,j+1,k}-f_{i,j-1,k}}{2l} \\
\frac{\partial f\left(x_i, y_j, z_k\right)}{\partial z} &\simeq \frac{f_{i,j,k+1}-f_{i,j,k-1}}{2m}.
\end{aligned} \tag{3.17}$$

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

$$\begin{aligned}
\frac{\partial^2 f\left(x_i, y_j, z_k\right)}{\partial x^2} &\simeq \frac{f\left(x_i-\Delta x, y_j, z_k\right)-2f\left(x_i, y_j, z_k\right)+f\left(x_i+\Delta x, y_j, z_k\right)}{\Delta x^2} \\
\frac{\partial^2 f\left(x_i, y_j, z_k\right)}{\partial y^2} &\simeq \frac{f\left(x_i, y_j-\Delta y, z_k\right)-f\left(x_i, y_j, z_k\right)+f\left(x_i, y_j+\Delta y, z_k\right)}{\Delta y^2} \\
\frac{\partial^2 f\left(x_i, y_j, z_k\right)}{\partial z^2} &\simeq \frac{f\left(x_i, y_j, z_k-\Delta z\right)-f\left(x_i, y_j, z_k\right)+f\left(x_i, y_j, z_k+\Delta z\right)}{\Delta z^2}
\end{aligned} \tag{3.18}$$

o en su forma simplificada tenemos

$$\begin{aligned}
\frac{\partial^2 f\left(x_i, y_j, z_k\right)}{\partial x^2} &\simeq \frac{f_{i-1,j,k}-2f_{i,j,k}+f_{i+1,j,k}}{h^2} \\
\frac{\partial^2 f\left(x_i, y_j, z_k\right)}{\partial y^2} &\simeq \frac{f_{i,j-1,k}-2f_{i,j,k}+f_{i,j+1,k}}{l^2} \\
\frac{\partial^2 f\left(x_i, y_j, z_k\right)}{\partial z^2} &\simeq \frac{f_{i,j,k-1}-2f_{i,j,k}+f_{i,j,k+1}}{m^2}.
\end{aligned} \tag{3.19}$$

4 Las Ecuaciones de Navier-Stokes

Las ecuaciones de Navier-Stokes reciben su nombre de los físicos Claude Louis Navier y George Gabriel Stokes (Francés e Irlandés respectivamente), quienes aplicaron los principios de la mecánica y termodinámica, resultando las ecuaciones en derivadas parciales no lineales que logran describir el comportamiento de un fluido. Estas ecuaciones no se concentran en una posición sino en un campo de velocidades, más específicamente en el flujo de dicho campo, lo cual es la descripción de la velocidad del fluido en un punto dado en el tiempo y en el espacio.

Cuando se estudia la dinámica de fluidos se consideran los siguientes componentes:

- $\overrightarrow{\mu}$, este término se usa para definir la velocidad del fluido
- ρ, este término se relaciona con la densidad del fluido
- p, este término hace referencia con la presión o fuerza por unidad de superficie que el fluido ejerce
- g, este término se relaciona con la aceleración de la gravedad, la cuál esta aplicada a todo el cuerpo, en determinados casos la gravedad no es la única fuerza ejercida sobre el cuerpo, por tal razón de toma como la sumatoria de fuerzas externas
- v, este término hace referencia a la viscosidad cinemática

La ecuación de Navier-Stokes general es

$$\rho\left(\frac{\partial v}{\partial t} + v \cdot \nabla v\right) = -\nabla p + \nabla \cdot \mathbb{T} + f \tag{4.1}$$

el término $\mathbb{T}$, expresa el tensor de estrés para los fluidos y condensa información acerca de las fuerzas internas del fluido. Supondremos al fluido incompresible cuyo campo de velocidades $\overrightarrow{\mu}$ satisface la condición $\nabla \cdot \overrightarrow{\mu} = 0$, por tal razón el tensor de estrés se reduce a $\nabla^2 \overrightarrow{\mu}$, dando como resultado la siguiente ecuación:

$$\frac{\partial \overrightarrow{\mu}}{\partial t} + \overrightarrow{\mu} \cdot \nabla \overrightarrow{\mu} + \frac{1}{\rho}\nabla p = g + v\nabla^2 \overrightarrow{\mu} \tag{4.2}$$

bajo la condición

$$\nabla \overrightarrow{\mu} = 0 \tag{4.3}$$

que garantiza la incompresibilidad del campo de velocidad sí la densidad permanece constante en el tiempo.

Así, las ecuaciones simplificadas de Navier-Stokes quedan como

$$\frac{\partial \overrightarrow{\mu}}{\partial t} = -\left(\overrightarrow{\mu}\nabla\right)\overrightarrow{\mu} + v\nabla^2 \overrightarrow{\mu} + f \tag{4.4}$$

$$\frac{\partial \rho}{\partial t} = -\left(\overrightarrow{\mu}\nabla\right)\rho + \kappa\nabla^2\rho + S \tag{4.5}$$

donde la primera ecuación describe la velocidad y la segunda hace referencia a la densidad a través de un campo vectorial. Estas ecuaciones son no lineales y no se conoce solución analítica a dicho problema.

4.1 La Solución Numérica de las Ecuaciones de Navier-Stokes

La solución numérica de las ecuaciones (4.4, 4.5) partiendo del estado inicial $\overrightarrow{\mu}_0$ del campo de velocidad y el primer instante de tiempo $t = 0$, se busca estudiar su comportamiento para el tiempo $t > 0$. Si $\overrightarrow{\omega}_0$ es la estimación del campo vectorial en el instante t y $\overrightarrow{\omega}_4$ denota el campo vectorial de velocidades en el instante de tiempo $t + \Delta t$. Jos Stam (véase [53]) propone una solución numérica a partir de la descomposición de la ecuación de distintos términos y solucionando cada uno individualmente. Así, la solución de cada paso se construye a partir del paso anterior. La solución en el tiempo $t + \Delta t$, está dada por el campo de velocidades $u\left(x, t + \Delta t\right) = \overrightarrow{\omega}_4$, la solución se obtiene iterando estos cuatro pasos:

1. Sumatoria de Fuerzas
2. Paso de Advección
3. Paso de Difusión
4. Paso de Proyección

i.e. $\overrightarrow{\omega}_0 \overset{1}{\overbrace{\rightarrow}} \overrightarrow{\omega}_1 \overset{2}{\overbrace{\rightarrow}} \overrightarrow{\omega}_2 \overset{3}{\overbrace{\rightarrow}} \overrightarrow{\omega}_3 \overset{4}{\overbrace{\rightarrow}} \overrightarrow{\omega}_4$

Sumatoria de Fuerzas La manera más práctica para incorporar la sumatoria de fuerzas externas f, se obtiene asumiendo que la fuerza no varía de manera considerable en un paso de tiempo. Bajo este supuesto, utilizando el método de Euler progresivo para resolver ecuaciones diferenciales ordinarias (teniendo en cuenta que la velocidad y la fuerza están relacionadas mediante la segunda ley de Newton, tenemos que

$$\overrightarrow{\omega}_1(x) = \overrightarrow{\omega}_0(x) + \Delta t f(x, t). \tag{4.6}$$

Paso de Advección Una perturbación en cualquier parte del fluido se propaga de acuerdo a la expresión

$$-\left(\overrightarrow{\mu} \cdot \nabla\right)\overrightarrow{\mu} \tag{4.7}$$

este término hace que la ecuación de Navier-Stokes sea no lineal, de aplicarse el método de Diferencias Finitas, éste es estable sólo cuando Δt sea suficientemente pequeño tal que $\Delta t < \Delta h / |u|$, donde Δh es el menor incremento de la malla

de discretización, para prevenir esto, se usa el método de Características. Este dice que una ecuación de la forma

$$\frac{\partial \alpha(x,t)}{\partial t} = -v(x)\nabla\alpha(x,t) \tag{4.8}$$

y

$$\alpha(x,t) = \alpha_0(x) \tag{4.9}$$

como las características del vector del campo v, que fluye a través del punto x_0 en $t = 0$, i.e.

$$\frac{d}{dt}p(x_0,t) = v(p(x_0,t)) \tag{4.10}$$

donde

$$p(x_0,0) = x_0. \tag{4.11}$$

De modo tal que $\alpha(x_0,t) = \alpha\left(p(x_0,t),t\right)$, es el valor del campo que pasa por el punto x_0 en $t = 0$. Para calcular la variación de esta cantidad en el tiempo se usa la regla de la cadena

$$\frac{d\alpha}{dt} = \frac{\partial\alpha}{\partial t} + v\nabla\alpha = 0 \tag{4.12}$$

que muestra que el valor de α no varía a lo largo de las líneas del flujo. En particular, se tiene que $\alpha(x_0,t) = \alpha(x_0,0) = \alpha_0(x_0)$; por lo tanto el campo inicial de las líneas de flujo, en el sentido inverso se pueden estimar el siguiente valor de α en x_0, esto es $\alpha\left(x_0, t+\Delta t\right)$.

Este método muestra en cada paso del tiempo, como las partículas del fluido se mueven por la velocidad del propio fluido. Por lo tanto, para obtener la velocidad en un punto x en el tiempo $t + \Delta t$, se devuelve al pinto x por el campo de velocidades $\overrightarrow{\omega}_1$ en el paso del tiempo Δt. Ésta define una ruta $p(x,s)$ correspondiente a la trayectoria parcial del campo de velocidades. La nueva velocidad en el punto x, es entonces la velocidad de la partícula ahora en x que tenía su anterior ubicación en el tiempo Δt. De esta manera, se puede hallar el valor de la velocidad luego de la advección $\overrightarrow{\omega}_2$ resolviendo el problema

$$\overrightarrow{\omega}_2(x) = \overrightarrow{\omega}_1(p(x,-\Delta t)) \tag{4.13}$$

la ventaja que se obtiene interpolando linealmente entre valores adyacentes de este método es su estabilidad numérica. Ya conocida la velocidad en el instante t, la viscosidad del fluido y su naturaleza obligan un proceso difusivo: La velocidad se propaga dentro del fluido, o bien en las partículas de este fluyen.

Paso de Difusión En este paso para resolver el efecto de la viscosidad y es equivalente a la ecuación de difusión

$$\frac{\partial\overrightarrow{\omega}_2(x)}{\partial t} = \nu\nabla^2\overrightarrow{\omega}_2(x) \tag{4.14}$$

esta es una ecuación para la cual se han desarrollado varios procedimientos numéricos. La forma más sencilla para resolver esta ecuación es discretizar el operador difusión ∇^2 y usar una forma explicita en el incremento del tiempo, sin embargo este método es inestable cuando la viscosidad es grande. Por ello una mejor opción es usar un método implícito para la ecuación

$$\left(\mathcal{I} - \nu\Delta t\nabla^2\right)\overrightarrow{\omega}_3(x) = \overrightarrow{\omega}_2(x) \tag{4.15}$$

donde $\mathcal{I}$ es el operador identidad.

Paso de Proyección Este último paso conlleva la proyección, que hace que resulte un campo libre de divergencia. Esto se logra mediante la solución del problema definido por

$$\nabla^2 q = \nabla \cdot \overrightarrow{\omega}_3(x) \qquad \text{y} \qquad \overrightarrow{\omega}_4(x) = \overrightarrow{\omega}3(x) - \nabla q \tag{4.16}$$

es necesario utilizar, según sea el caso, aquel método numérico que proporcione una solución correcta y eficiente a la ecuación de Poisson, esto es especialmente importante cuando hay vórtices en el fluido.

5 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

$$\underline{\underline{A}}\,\underline{u} = \underline{f} \tag{5.1}$$

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 librerías[17] para implementar de forma eficiente a los vectores, matrices —bandadas y dispersas— y resolver los sistemas lineales.

5.1 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.

5.1.1 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

[17] Algunas de las librerías más usadas para resolver sistemas lineales usando matrices bandadas y dispersar 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 $\underline{\underline{L}}^{-1}$ pueden obtenerse de los coeficientes $\underline{\underline{L}}_{ij}$ y pueden ser almacenados estrictamente en las entradas de la diagonal inferior de $\underline{\underline{A}}$ ya que estas ya fueron eliminadas. Esto proporciona una Factorización $\underline{\underline{LU}}$ de $\underline{\underline{A}}$ en la misma matriz $\underline{\underline{A}}$ 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 $\underline{\underline{A}}\underline{u} = \underline{f}$, a la matriz $\underline{\underline{A}}$, la cual será factorizada en la misma matriz, esta contendrá a las matrices $\underline{\underline{L}}$ y $\underline{\underline{U}}$ producto de la factorización, quedando el método numérico esquemáticamente como:

$$\begin{array}{l} \text{Para } i = 1, 2, ..., n \ \{ \\ \quad \text{Para } j = 1, 2, ..., n \ \{ \\ \quad\quad A_{ji} = A_{ji}/A_{ii} \\ \quad\quad \text{Para } k = i+1, ..., n \ \{ \\ \quad\quad\quad A_{jk} = A_{jk} - A_{ji}A_{ik} \\ \quad\quad \} \\ \quad \} \\ \} \end{array} \tag{5.2}$$

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

$$\underline{\underline{L}}\underline{y} = \underline{f} \quad \text{y} \quad \underline{\underline{U}}\underline{u} = \underline{y}. \tag{5.3}$$

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

$$\underline{\underline{L}}\underline{y} = \underline{f} \Leftrightarrow \begin{cases} y_1 = f_1/A_{11} \\ y_i = \left(f_i - \displaystyle\sum_{j=1}^{i-1} A_{ij} y_j \right) \text{ para toda } i = 2, ..., n \end{cases} \tag{5.4}$$

y

$$\underline{\underline{U}}\underline{u} = \underline{y} \Leftrightarrow \begin{cases} x_n = y_n/A_{nn} \\ u_i = \frac{1}{A_{ii}} \left(y_i - \displaystyle\sum_{j=i+1}^{n} A_{ij} x_j \right) \text{ para toda } i = n-1, ..., 1 \end{cases} . \tag{5.5}$$

La descomposición $\underline{\underline{LU}}$ requiere[18] $n^3/3 - n/3$ multiplicaciones/divisiones y $n^3/3 - n^2/2 + n/6$ sumas/restas —del orden $O(N^3/3)$ operaciones aritméticas

[18] Notemos que el método de eliminación gaussiana de una matriz $\underline{\underline{A}}$ 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 $\underline{\underline{L}}\underline{y} = \underline{f}$ y $\underline{\underline{U}}\underline{u} = \underline{y}$ requieren $n^2/2 - n/2$ operaciones aritméticas cada uno (véase [10] y [13]).

5.1.2 Factorización Cholesky

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

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

$$
\begin{array}{l}
A_{ii} = \sqrt{A_{11}} \\
\text{Para } j = 2, ..., n \text{ calcule } A_{j1} = A_{j1}/A_{11} \\
\text{Para } i = 2, ..., n \ \{ \\
\qquad A_{ii} = \sqrt{A_{ii} - \sum_{k=1}^{i-1} A_{ik}^2} \\
\qquad \text{Para } j = i+1, ..., n \\
\qquad\qquad A_{ji} = \left(A_{ji} - \sum_{k=1}^{i-1} A_{jk} A_{ik} \right) / A_{ii} \\
\} \\
A_{nn} = \sqrt{A_{nn} - \sum_{k=1}^{n-1} \left(A_{nk}\right)^2}
\end{array}
\tag{5.6}
$$

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

$$\underline{\underline{L}}\underline{y} = \underline{f} \quad \text{y} \quad \underline{\underline{L}}^T\underline{u} = \underline{y} \tag{5.7}$$

usando la formulación equivalente dada por las Ec.(5.4) y (5.5) para la descomposición LU.

La descomposición $\underline{\underline{LDL}}^T$ 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]).

5.1.3 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 $\underline{\underline{L}}$ y $\underline{\underline{U}}$ también se dejan en la matriz $\underline{\underline{A}}$ y después de la factorización, se resuelven los sistemas $\underline{\underline{L}}\underline{y} = \underline{f}$ y $\underline{\underline{U}}\underline{u} = \underline{y}$, quedando el método numérico esquemáticamente como:

$$
\begin{aligned}
&A_{12} = A_{12}/A_{11} \\
&y_1 = A_{1,n+1}/A_{11} \\
&\text{Para } i = 2, ..., n-1 \text{ \{} \\
&\qquad A_{ii} = A_{ii} - A_{i,i-1}A_{i-1,i} \\
&\qquad A_{i,i+1} = A_{i,i+1}/A_{ii} \\
&\qquad y_i = \left(A_{i,n+1} - A_{i,i-1}y_{i-1}\right)/A_{ii} \\
&\text{\}} \\
&A_{nn} = A_{nn} - A_{n,n-1}A_{n-1,n} \\
&y_n = \left(A_{n,n+1} - A_{n,n-1}y_{n-1}\right)/A_{nn} \\
&u_n = y_n \\
&\text{Para } i = n-1, ..., 1 \text{ tome } u_i = y_i - A_{i,i+1}u_{i+1}
\end{aligned}
\tag{5.8}
$$

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

5.2 Métodos Iterativos

En los métodos iterativos, se realizan iteraciones para aproximarse a la solución $\underline{u}$ aprovechando las características propias de la matriz $\underline{\underline{A}}$, 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

$$\underline{\underline{A}}\underline{u} = \underline{f} \tag{5.9}$$

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

$$\underline{u}^k = \underline{\underline{T}}\underline{u}^{k-1} + \underline{c} \quad \forall k = 1, 2, 3, ... \tag{5.10}$$

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

Teorema 7 *Si* $\left\|\underline{\underline{T}}\right\| < 1$*, entonces el sistema lineal* $\underline{u} = \underline{\underline{T}}\underline{u} + \underline{c}$ *tiene una solución única* $\underline{u}^*$ *y las iteraciones* $\underline{u}^k$ *definidas por la fórmula* $\underline{u}^k = \underline{\underline{T}}\underline{u}^{k-1} + \underline{c} \quad \forall k = 1, 2, 3, ...$ *convergen hacia la solución exacta* $\underline{u}^*$ *para cualquier aproximación inicial* $\underline{u}^0$.

Nótese que, mientras menor sea la norma de la matriz $\underline{\underline{T}}$, más rápida es la convergencia, en el caso cuando $\left\|\underline{\underline{T}}\right\|$ 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 $\left\|\underline{\underline{T}}\right\|$ 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 $\underline{\underline{A}}$, es por definición

$$cond(\underline{\underline{A}}) = \frac{\lambda_{\max}}{\lambda_{\min}} \geq 1 \tag{5.11}$$

donde $\lambda_{\max}$ y $\lambda_{\min}$ es el máximo y mínimo de los eigen-valores de la matriz $\underline{\underline{A}}$. 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 precondicionar 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 $h \to 0$ (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 [42] 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

$$\mathcal{K}_k = Generado\left\{\underline{r_0}, \underline{\underline{A}}\underline{r_0}, ..., \underline{\underline{A}}^{k-1}\underline{r_0}\right\} \text{ para } k \geq 1. \tag{5.12}$$

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

$$\underline{r_k} = \underline{b} - \underline{\underline{A}}\underline{x_k}. \tag{5.13}$$

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—.

5.2.1 Método de Gradiente Conjugado

Si la matriz generada por la discretización es simétrica —$\underline{\underline{A}} = \underline{\underline{A}}^T$— y definida positiva —$\underline{u}^T \underline{\underline{A}}\underline{u} > 0$ 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 $\mathcal{K}_n\left(\underline{\underline{A}}, \underline{v}^n\right)$ 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 Gramm-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 $\underline{\underline{A}}$ como simétrica y positiva definida, y como datos de entrada del sistema $\underline{\underline{A}}\underline{u} = \underline{f}$, el vector de búsqueda inicial $\underline{u}^0$ y se calcula $\underline{r}^0 = \underline{f} - \underline{\underline{A}}\underline{u}^0$, $\underline{p}^0 = \underline{r}^0$, quedando el método numérico esquemáticamente como:

$$
\begin{array}{l}
\alpha^n = \frac{\langle \underline{p}^n, \underline{p}^n \rangle}{\langle \underline{p}^n, \underline{\underline{A}}\underline{p}^n \rangle} \\
\underline{u}^{n+1} = \underline{u}^n + \alpha^n \underline{p}^n \\
\underline{r}^{n+1} = \underline{r}^n - \alpha^n \underline{\underline{A}}\underline{p}^n \\
\text{Prueba de convergencia} \\
\beta^n = \frac{\langle \underline{r}^{n+1}, \underline{r}^{n+1} \rangle}{\langle \underline{r}^n, \underline{r}^n \rangle} \\
\underline{p}^{n+1} = \underline{r}^{n+1} + \beta^n \underline{p}^n \\
n = n + 1
\end{array}
\tag{5.14}
$$

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 $\underline{\underline{A}}\underline{p}^n$, una manera de hacerlo queda esquemáticamente como:

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

Para $n = 1, 2, ...,$Mientras $(\mu < \varepsilon)$ {
$v = \underline{\underline{A}}\underline{p}$
$\alpha = \frac{\mu}{\underline{p} \cdot \underline{v}}$
$\underline{u} = \underline{u} + \alpha \underline{p}$

$\underline{r} = \underline{r} - \alpha \underline{v}$
$\mu' = \underline{r} \cdot \underline{r}$
$\beta = \frac{\mu'}{\mu}$
$\underline{p} = \underline{r} + \beta \underline{p}$
$\mu = \mu'$
}

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

$$\left\| \underline{u} - \underline{u}^k \right\|_{\underline{\underline{A}}} \leq \left\| \underline{u} - \underline{u}^0 \right\|_{\underline{\underline{A}}} \left[\frac{1 - \sqrt{Cond(\underline{\underline{A}})}}{1 + \sqrt{Cond(\underline{\underline{A}})}} \right]^{2k} . \tag{5.15}$$

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

Teorema 8 *Sea* $\kappa = cond(\underline{\underline{A}}) = \frac{\lambda_{\max}}{\lambda_{\min}} \geq 1$, *entonces el método de Gradiente Conjugado satisface la* $\underline{\underline{A}}-$*norma del error dado por*

$$\frac{\|e^n\|}{\|e^0\|} \leq \frac{2}{\left[\left(\frac{\sqrt{\kappa}+1}{\sqrt{\kappa}-1} \right)^n + \left(\frac{\sqrt{\kappa}+1}{\sqrt{\kappa}-1} \right)^{-n} \right]} \leq 2 \left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1} \right)^n \tag{5.16}$$

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

Nótese que para κ grande se tiene que

$$\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1} \simeq 1 - \frac{2}{\sqrt{\kappa}} \tag{5.17}$$

tal que

$$\|\underline{e}^n\|_{\underline{\underline{A}}} \simeq \left\| \underline{e}^0 \right\|_{\underline{\underline{A}}} \exp\left(-2 \frac{n}{\sqrt{\kappa}} \right) \tag{5.18}$$

de lo anterior se puede esperar un espectro de convergencia del orden de $O(\sqrt{\kappa})$ iteraciones (véase [14] y [42]).

Definición 9 *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 10 *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.*

5.2.2 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

$$\left\{\underline{v}^1, \underline{v}^2, ..., \underline{v}^n\right\} \tag{5.19}$$

para el espacio de Krylov $\mathcal{K}_n\left(\underline{\underline{A}}, \underline{v}^n\right)$. Para hacer $\underline{v}^{n+1}$ ortogonal a $\mathcal{K}_n\left(\underline{\underline{A}}, \underline{v}^n\right)$, es necesario usar todos los vectores previamente construidos $\left\{\underline{v}^{n+1j}\right\}_{j=1}^{n}$ —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 $\underline{\underline{V}}_n = \left[\underline{v}^1, \underline{v}^2, ..., \underline{v}^n\right]$ la cual denota la matriz conteniendo $\underline{v}^j$ en la j-ésima columna, para $j = 1, 2, ..., n$, y sea $\underline{\underline{H}}_n = \left[h_{i,j}\right], 1 \leq i, j \leq n$, donde las entradas de $\underline{\underline{H}}_n$ no especificadas en el algoritmo son cero. Entonces, $\underline{\underline{H}}_n$ es una matriz superior de Hessenberg. i.e. $h_{ij} = 0$ para $j < i - 1$, y

$$\begin{aligned} \underline{\underline{AV}}_n &= \underline{\underline{V}}_n \underline{\underline{H}}_n + h_{n+1,n}\left[0, ..., 0, \underline{v}^{n+1}\right] \\ \underline{\underline{H}}_n &= \underline{\underline{H}}_n^T \underline{\underline{AV}}_n. \end{aligned} \tag{5.20}$$

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

$$\underline{\underline{A}}\underline{u} = \underline{f} \tag{5.21}$$

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

$$\begin{array}{l} \text{Para } n = 1, 2, ..., \text{Mientras } \beta^n < \tau\beta^0 \ \{ \\ \quad \underline{w}_0^{n+1} = \underline{\underline{A}}\underline{v}^n \\ \quad \text{Para } l = 1 \text{ hasta } n \ \{ \\ \qquad h_{l,n} = \left\langle \underline{w}_l^{n+1}, \underline{v}^l \right\rangle \\ \qquad \underline{w}_{l+1}^{n+1} = \underline{w}_l^{n+1} - h_{l,n}\underline{v}^l \\ \quad \} \\ \quad h_{n+1,n} = \left\|\underline{w}_{n+1}^{n+1}\right\| \\ \quad \underline{v}^{n+1} = \underline{w}_{n+1}^{n+1}/h_{n+1,n} \\ \quad \text{Calcular } \underline{y}^n \text{ tal que } \beta^n = \left\|\beta^0\underline{e}_1 - \underline{\underline{\hat{H}}}_n\underline{y}^n\right\| \text{ es mínima} \\ \} \end{array} \tag{5.22}$$

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

$$\underline{r}^n = \underline{r}^0 - \underline{\underline{AV}}_n y^n = \underline{\underline{V}}_{n+1}\left(\beta^0\underline{e}_1 - \underline{\underline{\hat{H}}}_n\underline{y}^n\right). \tag{5.23}$$

Teorema 11 *Sea $\underline{u}^k$ la iteración generada después de k iteraciones de GMRES, con residual $\underline{r}^k$. Si la matriz $\underline{\underline{A}}$ es diagonalizable, i.e. $\underline{\underline{A}} = \underline{\underline{V}}\underline{\underline{\Lambda}}\underline{\underline{V}}^{-1}$ donde Λ es una matriz diagonal de eigen-valores de $\underline{\underline{A}}$, y $\underline{\underline{V}}$ es la matriz cuyas columnas son los eigen-vectores, entonces*

$$\frac{\left\|r^k\right\|}{\left\|r^0\right\|} \leq \kappa\left(V\right) \min_{p_\kappa \in \Pi\kappa, p_k(0)=1} \max_{\lambda_j} \left|p_\kappa\left(\lambda_j\right)\right| \tag{5.24}$$

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

5.3 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 almacenar[19], 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 $\underline{\underline{A}}\underline{u} = \underline{f}$, donde la matriz $\underline{\underline{A}}$ —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 $\underline{v} = \underline{\underline{C}}\underline{u}$, 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 Cache[20] del procesador.

[19] En el caso de los ejemplos de la sección (6) 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.

[20] Nótese 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 $\underline{\underline{C}}$ 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 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.

5.3.1 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

$$\underline{\underline{A}} = \begin{bmatrix} a_1 & b_1 & & & c_1 & & & & \\ d_2 & a_2 & b_2 & & & c_2 & & & \\ & d_3 & a_3 & b_3 & & & c_3 & & \\ & & d_4 & a_4 & b_4 & & & c_4 & \\ e_5 & & & d_5 & a_5 & b_5 & & & c_5 \\ & e_6 & & & d_6 & a_6 & b_6 & & \\ & & e_7 & & & d_7 & a_7 & b_7 & \\ & & & e_8 & & & d_8 & a_8 & b_8 \\ & & & & e_9 & & & d_9 & a_9 \end{bmatrix} \tag{5.25}$$

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

$$\underline{Ind} = [-5, -1, 0, +1, +5] \tag{5.26}$$

y los datos serán almacenados usando la estructura

$$\underline{\underline{Dat}} = \begin{bmatrix} 0 & 0 & a_1 & b_1 & c_1 \\ 0 & d_2 & a_2 & b_2 & c_2 \\ 0 & d_3 & a_3 & b_3 & c_3 \\ 0 & d_4 & a_4 & b_4 & c_4 \\ e_5 & d_5 & a_5 & b_5 & c_5 \\ e_6 & d_6 & a_6 & b_6 & 0 \\ e_7 & d_7 & a_7 & b_7 & 0 \\ e_8 & d_8 & a_8 & b_8 & 0 \\ e_9 & d_9 & a_9 & 0 & 0 \end{bmatrix} \tag{5.27}$$

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 $\underline{Ind}$ —supóngase en la columna k—, entonces $A_{i,j} = Dat_{ik}$, 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.

5.3.2 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

$$\underline{\underline{A}} = \begin{bmatrix} a_1 & & & & & b_1 & & & c_1 \\ & a_2 & & & b_2 & & & c_2 & \\ & & & & a_3 & & & b_3 & c_3 \\ a_4 & & & b_4 & & & & & \\ & & a_5 & & & b_5 & & & c_5 \\ a_6 & b_6 & c_6 & & & & & & \\ & & & & & & a_7 & b_7 & c_7 \\ & & & a_8 & & & b_8 & c_8 & \\ & & & & & & a_9 & b_9 & \end{bmatrix} \tag{5.28}$$

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

$$\underline{\underline{Ind}} = \begin{bmatrix} 1 & 6 & 9 \\ 2 & 5 & 8 \\ 5 & 8 & 9 \\ 1 & 4 & 0 \\ 3 & 6 & 9 \\ 1 & 2 & 3 \\ 7 & 8 & 9 \\ 4 & 7 & 8 \\ 7 & 8 & 0 \end{bmatrix} \tag{5.29}$$

y los datos serán almacenados usando la estructura

$$\underline{\underline{Dat}} = \begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \\ a_4 & b_4 & 0 \\ a_5 & b_5 & c_5 \\ a_6 & b_6 & c_6 \\ a_7 & b_7 & c_7 \\ a_8 & b_8 & c_8 \\ a_9 & b_9 & 0 \end{bmatrix} \tag{5.30}$$

de tal forma que la matriz $\underline{\underline{A}}$ puede ser reconstruida de forma eficiente. Para obtener el 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 $A_{i,j} = Dat_{ik}$, en otro caso $A_{i,j} = 0$.

Casos Particulares de la Matriz Dispersa $\underline{\underline{A}}$ Si la matriz $\underline{\underline{A}}$, 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 $r << n$, entonces es posible sólo guardar los r renglones distintos y llevar un arreglo que contenga la referencia al renglón almacenado.

5.3.3 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. $\underline{\underline{C}}\underline{u}$ 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 $\underline{\underline{C}}$ 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.

6 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 implementación[21] en los paquetes de cómputo OCTAVE (MatLab), SciLab y en los lenguajesde programación C++, Python, Java, Mono (C#), Fortran y C, estos ejemplos y el presente texto se pueden descargar de la página WEB:

http://mmc.geofisica.unam.mx/acl/MDF/

o desde GitHub[22] mediante

git clone git://github.com/antoniocarrillo69/MDF.git

6.1 Implementación en Octave (MatLab)

GNU OCTAVE[23] es un paquete de cómputo open source para el cálculo numérico —muy parecido a MatLab[24] pero sin costo alguno para el usuario— el cual provee un poderoso ambiente de cálculo para aplicaciones Científicas y de Ingeniería.

Ejemplo 15 *Sea*

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

entonces el programa queda implementado como:

```
function [A,b,x] = fdm1d(n)
    xi = -1; % Inicio de dominio
    xf = 2; % Fin de dominio
    vi = -1; % Valor en la forntera xi
    vf = 1; % Valor en la frontera xf
    N=n-2; % Nodos interiores
    h=(xf-xi)/(n-1); % Incremento en la malla
    A=zeros(N,N); % Matriz A
    b=zeros(N,1); % Vector b
```

[21] En los ejemplos cuando es posible se definen matrices que minimizan la memoria usada en el almacenamiento y maximizan la eficiencia de los métodos numéricos de solución del sistema lineal asociado. En el caso de Octave (MatLab) se define a una matriz mediante A = zeros(N,N), esta puede ser remplazada por una matriz que no guarda valores innecesarios (ceros), esto se hace mediante la declaración de la matriz como A = sparse(N,N).

[22] GitHub es un plataforma de desarrollo colaborativo para alojar proyectos usando el sistema de control de versiones GIT.

[23] GNU Octave [http://www.gnu.org/software/octave/]

[24] MatLab [http://www.mathworks.com/products/matlab/]

```
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(xi)-vi*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(xi+h*(i-1));
end
% Relglon final de la matriz A y vector b
A(N,N-1)=R;
A(N,N)=P;
b(N)=LadoDerecho(xi+h*N)-vf*Q;
% Resuleve el sistema lineal Ax=b
x=inv(A)*b;
% Prepara la graficacion
xx=zeros(n,1);
zz=zeros(n,1);
for i=1:n
xx(i)=xi+h*(i-1);
zz(i)=SolucionAnalitica(xx(i));
end
yy=zeros(n,1);
yy(1)=vi; % Condicion inicial
for i=1:N
      yy(i+1)=x(i);
end
yy(n)=vf; % Condicion inicial
% Grafica la solucion de la Ecuacion Diferencial Parcial en 1D
plot(xx,[yy,zz]);
%plot(xx,zz);
endfunction
% Lado derecho de la ecuacion
function y=LadoDerecho(x)
 y=-pi*pi*cos(pi*x);
endfunction
% Solucion analitica a la ecuacion
function y=SolucionAnalitica(x)
 y=cos(pi*x);
endfunction
```

Si el programa lo grabamos en el directorio de trabajo con el nombre **fdm1d.m**, entonces se puede ejecutar en la consola de OCTAVE (MatLab) mediante

$$[A, b, x] = fdm1d(30);$$

donde es necesario indicar el número de nodos (30) en la partición. La ejecución genera la gráfica de la solución calculada por el método de diferencias finitas y la solución analítica en los mismos puntos de la malla, además devuelve la matriz y los vectores A, b, x generados por la función.

Dado que esta escuación se puede extender a todo es espacio, entonces se puede reescribir así

Ejemplo 16 *Sea*

$$-u''(x) = -\pi^2 \cos(\pi x), \qquad xi \leq x \leq xf, \qquad u(xi) = vi, \qquad u(xf) = xf$$

entonces el programa queda implementado como:

```
function [A,b,x] = fdm1d(xi,xf,vi,vf,n)
  N=n-2; % Nodos interiores
  h=(xf-xi)/(n-1); % Incremento en la malla
  A=zeros(N,N); % Matriz A
  b=zeros(N,1); % 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(xi)-vi*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(xi+h*(i-1));
  end
  % Renglon final de la matriz A y vector b
  A(N,N-1)=R;
  A(N,N)=P;
  b(N)=LadoDerecho(xi+h*N)-vf*Q;
  % Resuleve el sistema lineal Ax=b
  x=inv(A)*b;
  % Prepara la graficacion
  xx=zeros(n,1);
  zz=zeros(n,1);
  for i=1:n
```

```
    xx(i)=xi+h*(i-1);
    zz(i)=SolucionAnalitica(xx(i));
  end
  yy=zeros(n,1);
  yy(1)=vi; % Condicion inicial
  for i=1:N
    yy(i+1)=x(i);
  end
  yy(n)=vf; % Condicion inicial
  % Grafica la solucion de la Ecuacion Diferencial Parcial en 1D
  plot(xx,[yy,zz]);
endfunction
function y=LadoDerecho(x)
  y=-pi*pi*cos(pi*x);
endfunction
function y=SolucionAnalitica(x)
  y=cos(pi*x);
endfunction
```

Si el programa lo grabamos en el directorio de trabajo con el nombre **fdm1d.m**, entonces se puede ejecutar en la consola de OCTAVE (MatLab) mediante

$$[A, b, x] = fdm1d(-1, 2, -1, 1, 30);$$

donde es necesario indicar el inicio (-1) y fin (2) del dominio, el valor de la condición de frontera de inicio (-1) y fin (1), además de el número de nodos (30) en la partición. La ejecución genera la gráfica de la solución calculada por el método de diferencias finitas y la solución analítica en los mismos puntos de la malla, además devuelve la matriz y los vectores A, b, x generados por la función.

6.2 Implementación en SciLab

Scilab[25] 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 17 *Sea*

$$-u''(x) = -\pi^2 \cos(\pi x), \qquad xi \leq x \leq xf, \qquad u(xi) = vi, \qquad u(xf) = vf$$

El programa queda implementado como:

```
function [A,b,x] = fdm1d(n)
    xi = -1; // Inicio de dominio
    xf = 2; // Fin de dominio
    vi = -1; // Valor en la forntera xi
    vf = 1; // Valor en la frontera xf
    N=n-2; // Nodos interiores
    h=(xf-xi)/(n-1); // Incremento en la malla
    A=zeros(N,N); // Matriz A
    b=zeros(N,1); // 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(xi)-vi*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(xi+h*(i-1));
    end
    // Relglon final de la matriz A y vector b
    A(N,N-1)=R;
    A(N,N)=P;
    b(N)=LadoDerecho(xi+h*N)-vf*Q;
    // Resuleve el sistema lineal Ax=b
    x=inv(A)*b;
    // Prepara la graficacion
    xx=zeros(n,1);
    zz=zeros(n,1);
    for i=1:n
        xx(i)=xi+h*(i-1);
```

[25] Scilab [http://www.scilab.org]

```
        zz(i)=SolucionAnalitica(xx(i));
    end
    yy=zeros(n,1);
    yy(1)=vi; // Condicion inicial
    for i=1:N
        yy(i+1)=x(i);
    end
    yy(n)=vf; // Condicion inicial
    // Grafica la solucion de la Ecuacion Diferencial Parcial en 1D
    plot2d(xx,[yy,zz]);
endfunction
// Lado derecho de la ecuacion
function y=LadoDerecho(x)
    y=-%pi*%pi*cos(%pi*x);
endfunction
// Solucion analitica a la ecuacion
function y=SolucionAnalitica(x)
    y=cos(%pi*x);
endfunction
```

Si el programa lo grabamos en el directorio de trabajo con el nombre **fdm1d.sci**, entonces para poder ejecutar la función es necesario cargarla en la consola de SCILAB mediante

exec('./fdm1d.sci',-1)

para después ejecutar la función mediante:

$$[A, b, x] = fdm1d(30);$$

donde es necesario indicar el número de nodos (30) en la partición. La ejecución genera la gráfica de la solución calculada por el método de diferencias finitas y la solución analítica en los mismos puntos de la malla, además devuelve la matriz y los vectores A, b, x generados por la función.

Dado que esta escuación se puede extender a todo es espacio, entonces se puede reescribir así

Ejemplo 18 *Sea*

$$-u''(x) = -\pi^2 \cos(\pi x), \qquad xi \leq x \leq xf, \qquad u(xi) = vi, \qquad u(xf) = vf$$

El programa queda implementado como:

```
function [A,b,x] = fdm1d(xi,xf,vi,vf,n)
   N=n-2; // Nodos interiores
   h=(xf-xi)/(n-1); // Incremento en la malla
   A=zeros(N,N); // Matriz A
   b=zeros(N,1); // 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(xi)-vi*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(xi+h*(i-1));
  end
  // Renglon final de la matriz A y vector b
  A(N,N-1)=R;
  A(N,N)=P;
  b(N)=LadoDerecho(xi+h*N)-vf*Q;
  // Resuleve el sistema lineal Ax=b
  x=inv(A)*b;
  // Prepara la graficacion
  xx=zeros(n,1);
  zz=zeros(n,1);
  for i=1:n
    xx(i)=xi+h*(i-1);
    zz(i)=SolucionAnalitica(xx(i));
  end
  yy=zeros(n,1);
  yy(1)=vi; // Condicion inicial
  for i=1:N
    yy(i+1)=x(i);
  end
  yy(n)=vf; // Condicion inicial
  // Grafica la solucion de la Ecuacion Diferencial Parcial en 1D
  plot2d(xx,[yy,zz]);
endfunction
function y=LadoDerecho(x)
  y=-%pi*%pi*cos(%pi*x);
endfunction
function y=SolucionAnalitica(x)
  y=cos(%pi*x);
endfunction
```

Si el programa lo grabamos en el directorio de trabajo con el nombre **fdm1d.sci**, entonces para poder ejecutar la función es necesario cargarla en la consola de SCILAB mediante

exec('./fdm1d.sci',-1)

para después ejecutar la función mediante:

$$[A, b, x] = fdm1d(-1, 2, -1, 1, 30);$$

donde es necesario indicar el inicio (-1) y fin (2) del dominio, el valor de la condición de frontera de inicio (-1) y fin (1), además de el numero de nodos (30) en la partición. La ejecución genera la gráfica de la solución calculada por el método de diferencias finitas y la solución analítica en los mismos puntos de la malla, además devuelve la matriz y los vectores A, b, x generados por la función.

Observación 1 *La diferencia entre los códigos de OCTAVE y SCILAb al menos para estos ejemplos estriba en la forma de indicar los comentarios y la manera de llamar para generar la gráfica, además de que en SCILAB es necesario cargar el archivo que contiene la función.*

Ejemplo 19 *Sea*

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

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);

// 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
// Relglon 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

// Resuleve 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 20 *Sea*

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

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
 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 21 *Sea*

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

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
// Relglon 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;

// Resuleve 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])
```

Ejemplo 22 *Sea*

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

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);

// 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
// Relglon 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

// Resuleve 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])
```

6.3 Implementación en C++

GMM++[26] 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++ ejemplito.cpp
```

Para ejecutar usar:

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

Ejemplo 23 *Sea*

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

entonces el programa queda implementado como:

```
#include <gmm/gmm.h>
#include <math.h>
const double pi = 3.141592653589793;
// Lado derecho
double LD(double x)
{
return ( -pi * pi * cos(pi * x));
}
// Solucion analitica
double SA(double x)
{
return (cos(pi * x));
}
int main(void)
{
int M=11; // Particion
int N=M-2; // Nodos interiores
double a=0; // Inicio dominio
double c=1; // Fin dominio
double h=(c-a)/(M-1); // Incremento en la malla
double Y0=1.0; // 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
```

[26] GMM++ [http://download.gna.org/getfem/html/homepage/gmm/]

```
std::vector<double> x(N), b(N);
int i;
double P = -2 / (h * h);
double Q = 1 / (h * h);
double R = 1 / (h * h);
A(0, 0) = P; // Primer renglon de la matriz A y vector b
A(0, 1) = Q;
b[0] = LD(a + h) - (Y0 / (h * h));
// 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;
b[i] = LD(a + (i + 1) * h);
}
A(N - 1, N - 2) = R; // Relglon final de la matriz A y vector b
A(N - 1, N - 1) = P;
b[N - 1] = LD(a + (i + 1) * h) - (Y1 / (h * h));
// Copia la matriz dispersa a la densa para usarla en LU
gmm::copy(A,AA);
// Visualiza la matriz y el vector
std::cout << "Matriz A"<< AA << gmm::endl;
std::cout << "Vector b"<< b << gmm::endl;
// 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
// Conjugate gradient
gmm::cg(A, x, b, PS, PR, iter);
std::cout << "CGM"<< x << std::endl;
// BICGSTAB BiConjugate Gradient Stabilized
gmm::bicgstab(A, x, b, PR, iter);
std::cout << "BICGSTAB"<< x << std::endl;
// GMRES generalized minimum residual
gmm::gmres(A, x, b, PR, restart, iter);
std::cout << "GMRES"<< x << std::endl;
// Quasi-Minimal Residual method
gmm::qmr(A, x, b, PR, iter);
std::cout << "Quasi-Minimal"<< x << std::endl;
// Visualiza la solucion numerica
std::cout << "Solucion Numerica"<< std::endl;
std::cout << a << " " << Y0 << gmm::endl;
```

```
for(i = 0; i < N; i++)
{
std::cout << (i + 1)*h << " " << x[i] << gmm::endl;
}
std::cout << c << " " << Y1 << gmm::endl;
// Visualiza la solucion analitica
std::cout << "Solucion Analitica"<< std::endl;
std::cout << a << " " << SA(a) << gmm::endl;
for(i = 0; i < N; i++)
{
std::cout << (i + 1)*h << " " << SA((a + (i + 1)*h)) << gmm::endl;
}
std::cout << c << " " << SA(c) << gmm::endl;
// Visualiza el error en valor absoluto en cada nodo
std::cout << "Error en el calculo"<< std::endl;
std::cout << a << "      " << abs(Y0 - SA(a)) << gmm::endl;
for(i = 0; i < N; i++)
{
std::cout << (i + 1)*h << "      " << abs(x[i] - SA((a + (i + 1)*h))) <<
gmm::endl;
}
std::cout << c << "      " << abs(Y1 - SA(c)) << gmm::endl;
return 0;
}
```

6.4 Implementación en Python

Python[27] es un lenguaje de programación interpretado, usa tipado dinámico y es multiplataforma cuya filosofía hace hincapié en una sintaxis que favorezca el código legible y soporta programación multiparadigma —soporta orientación a objetos, programación imperativa y programación funcional—, además es desarrollado bajo licencia de código abierto.

Ejemplo 24 *Sea*

$u_t + a(x)u'(x) = 0,$ *si* $u(x,0) = f(x),$ *la solución analítica es* $u(x,t) = f(x-at)$

entonces el programa queda implementado como:

```
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
```

[27] Python [http://www.python.org]

```
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 guradarÃ¡ la soluciÃ³n 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 aproximaciÃ³n"
else:
print "mu ", mu
print "Mala aproximaciÃ³n"
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 soulción analitica
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
#for k_ in xrange (nNodos):
# xx[k_] = sol_ analitica(espacio[k_], t)
# print xx
plt.plot(espacio, w)
#plt.plot(espacio, xx)
# print "XX"
# print xx
# print "W"
# print w
```

6.5 Implementación en Java

Java es un lenguaje de programación orientado a objetos de propósito general. concurrente y multiplataforma desarrollado bajo licencia de código abierto. El lenguaje no está diseñado especificamente para cómputo científico, pero es facil implementar lo necesario para nuestros fines.

Sea

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

Ejemplo 25 *El programa queda implementado en JAVA como:*

```
public class fdm1d {
      private int Visualiza;
```

```
// Constructor
public fdm1d() {
      Visualiza = 1;
}
// Resuelve Ax=b usando el metodo Jacobi
public void Jacobi(double[][] A, double[] x, double[] b, int n, int iter) {
      int i, j, m;
      double sum;
      double[] xt = new double [n];
     for (m = 0; m < iter; m ++) {
           for (i = 0; i < n; i++) {
                 sum = 0.0;
                 for (j = 0; j < n; j ++) {
                       if ( i == j) continue;
                            sum += A[i][j] * x[j];
                 }
                 if (A[i][i] == 0.0) return;
                 xt[i] = (1.0 / A[i][i]) * (b[i] - sum);
           }
           for (i = 0; i < n; i++) x[i] = xt[i];
      }
      if (Visualiza != 0) {
           System.out.println(" ");
           System.out.println("Matriz");
           for (i = 0; i < n; i++) {
                 for (j = 0; j < n; j++) {
                       System.out.print(A[i][j] + " ");
                 }
                 System.out.println(" ")
           }
           System.out.println(" ");
           System.out.println("b");
           for (i = 0; i < n; i++) {
                 System.out.print(b[i] + " ");
           }
           System.out.println(" ");
           System.out.println("x");
           for (i = 0; i < n; i++) {
                 System.out.print(x[i] + " ");
           }
           System.out.println(" ");
      }
}
// Resuelve Ax=b usando el metodo Gauus-Siedel
public void Gauss_Siedel(double[][] A, double[] x, double[] b, int n, int
iter) {
```

```
        int i, j, m;
        double sum;
       for (m = 0; m < iter; m ++) {
            for (i = 0; i < n; i++) {
                sum = 0.0;
                for (j = 0; j < n; j ++) {
                    if ( i == j) continue;
                    sum += A[i][j] * x[j];
                }
                if (A[i][i] == 0.0) return;
               x[i] = (1.0 / A[i][i]) * (b[i] - sum);
           }
        }
        if (Visualiza != 0) {
            System.out.println(" ");
            System.out.println("Matriz");
            for (i = 0; i < n; i++) {
                for (j = 0; j < n; j++) {
                    System.out.print(A[i][j] + " ");
                }
                System.out.println(" ");
            }
            System.out.println(" ");
            System.out.println("b");
            for (i = 0; i < n; i++) {
                System.out.print(b[i] + " ");
            }
            System.out.println(" ");
            System.out.println("x");
            for (i = 0; i < n; i++) {
                System.out.print(x[i] + " ");
            }
            System.out.println(" ");
        }
    }
    // Lado derecho de la ecuacion diferencial parcial
    public static double LadoDerecho(double x) {
        double pi = 3.14159265358979323846264338332;
        return -pi * pi * java.lang.Math.cos(pi * x);
    }
    // Funcion Principal ....
    public static void main(String[] args) {
        double xi = -1.0; // Inicio del diminio
        double xf = 2.0; // Fin del dominio
        double vi = -1.0; // Valor en la frontera xi
        double vf = 1.0; // Valor en la frontera xf
```

```
            int n = 11; // Particion
            int N = n - 2; // Nodos interiores
            double h = (xf - xi) / (n - 1); // Incremento en la malla
            double[][] A = new double[N][N]; // Matriz A
            double[] b = new double[N]; // Vector b
            double[] x = new double[N]; // Vector x
            double R = 1 / (h * h);
            double P = -2 / (h * h);
            double Q = 1 / (h * h);
            // Primer renglon de la matriz A y vector b
            A[0][0] = P;
            A[0][1] = Q;
            b[0] = LadoDerecho(xi) - vi * R;
            // Renglones intermedios de la matriz A y vector b
            for (int i = 1; i < N - 1; i++) {
                A[i][i - 1] = R;
                A[i][i] = P;
                A[i][i + 1] = Q;
                b[i] = LadoDerecho(xi + h * i);
            }
            // Renglon final de la matriz A y vector b
            A[N - 1][N - 2] = R;
            A[N - 1][N - 1] = P;
            b[N - 1] = LadoDerecho(xi + h * N - 2) - vf * Q;
            // Resuleve el sistema lineal Ax=b
            fdm1d ejem = new fdm1d();
            ejem.Gauss_Siedel(A, x, b, N, 1000);
            ejem.Jacobi(A, x, b, N, 1000);
        }
    }
```

6.6 Implementación en MONO (C#)

Mono (C#) es un lenguaje de programación orientado a bojetos creado para desarrollar un grupo de herramientas libres basadas en GNU/Linux y compatibles con .NET (C#), no está especialmente desarrollado para cómputo científico, pero es facil implementar lo necesario para nuestros fines.

Sea

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

Ejemplo 26 *El programa queda implementado en MONO como:*

```
using System;
using System.IO;
using System.Text;
namespace FDM1D
```

```
{
public class fdm1d {
        private int Visualiza;
        // Constructor
        public fdm1d() {
              Visualiza = 1;
        }
              // Resuelve Ax=b usando el metodo Jacobi
             public void Jacobi(double[,] A, double[] x, double[] b, int n, int iter)
{
              int i, j, m;
              double sum;
                   double[] xt = new double [n];
                   for (m = 0; m < iter; m ++) {
                         for (i = 0; i < n; i++) {
                          sum = 0.0;
                          for (j = 0; j < n; j ++) {
                                if ( i == j) continue;
                                sum += A[i,j] * x[j];
                          }
                          if (A[i,i] == 0.0) return;
                          xt[i] = (1.0 / A[i,i]) * (b[i] - sum);
                    }
                    for (i = 0; i < n; i++) x[i] = xt[i];
              }
              if (Visualiza != 0) {
                    Console.WriteLine(" ");
                    Console.WriteLine("Matriz");
                    for (i = 0; i < n; i++) {
                          for (j = 0; j < n; j++) {
                                System.Console.Write(A[i,j] + " ");
                          }
                          System.Console.WriteLine(" ");
                    }
                    System.Console.WriteLine(" ");
                    System.Console.WriteLine("b");
                    for (i = 0; i < n; i++) {
                          System.Console.Write(b[i] + " ");
                    }
                    System.Console.WriteLine(" ");
                    System.Console.WriteLine("x");
                    for (i = 0; i < n; i++) {
                          System.Console.Write(x[i]+ " ");
                    }
                    System.Console.WriteLine(" ");
              }
```

```
        }
        // Resuelve Ax=b usando el metodo Gauus-Siedel
        public void Gauss_ Siedel(double[,] A, double[] x, double[] b, int n, int
iter) {
            int i, j, m;
            double sum;
            for (m = 0; m < iter; m ++) {
                for (i = 0; i < n; i++) {
                    sum = 0.0;
                    for (j = 0; j < n; j ++) {
                        if ( i == j) continue;
                        sum += A[i,j] * x[j];
                    }
                    if (A[i,i] == 0.0) return;
                    x[i] = (1.0 / A[i,i]) * (b[i] - sum);
                }
            }
            if (Visualiza != 0) {
                Console.WriteLine(" ");
                Console.WriteLine("Matriz");
                for (i = 0; i < n; i++) {
                    for (j = 0; j < n; j++) {
                        System.Console.Write(A[i,j] + " ");
                    }
                    System.Console.WriteLine(" ");
                }
                System.Console.WriteLine(" ");
                System.Console.WriteLine("b");
                for (i = 0; i < n; i++) {
                    System.Console.Write(b[i] + " ");
                }
                System.Console.WriteLine(" ");
                System.Console.WriteLine("x");
                for (i = 0; i < n; i++) {
                    System.Console.Write(x[i] + " ");
                }
                System.Console.WriteLine(" ");
            }
        }
        // Lado derecho de la ecuacion diferencial parcial
        public static double LadoDerecho(double x) {
            double pi = 3.14159265358979323846264338 32;
            return -pi * pi * Math.Cos(pi * x);
        }
        // Funcion Principal ....
        public static void Main(String[] args) {
```

```
        double xi = -1.0; // Inicio del diminio
        double xf = 2.0; // Fin del dominio
        double vi = -1.0; // Valor en la frontera xi
        double vf = 1.0; // Valor en la frontera xf
        int n = 11; // Particion
        int N = n - 2; // Nodos interiores
        double h = (xf - xi) / (n - 1); // Incremento en la malla
        double[,] A = new double[N,N]; // Matriz A
        double[] b = new double[N]; // Vector b
        double[] x = new double[N]; // Vector x
        double R = 1 / (h * h);
        double P = -2 / (h * h);
        double Q = 1 / (h * h);
        // Primer renglon de la matriz A y vector b
        A[0,0] = P;
        A[0,1] = Q;
        b[0] = LadoDerecho(xi) - vi * R;
        // Renglones intermedios de la matriz A y vector b
        for (int i = 1; i < N - 1; i++) {
            A[i,i - 1] = R;
            A[i,i] = P;
            A[i,i + 1] = Q;
            b[i] = LadoDerecho(xi + h * i);
        }
        // Renglon final de la matriz A y vector b
        A[N - 1,N - 2] = R;
        A[N - 1,N - 1] = P;
        b[N - 1] = LadoDerecho(xi + h * N - 2) - vf * Q;
        // Resuleve el sistema lineal Ax=b
        fdm1d ejem = new fdm1d();
        ejem.Gauss_Siedel(A, x, b, N, 1000);
        ejem.Jacobi(A, x, b, N, 1000);
    }
}
}
```

6.7 Implementación en Fortran

Fortran es un lenguaje de programación procedimental e imperativo que está adaptado al cálculo numérico y la computación científica, existen una gran variedad de subrutinas para manipulación de matrices, pero es facil implementar lo necesario para nuestros fines.

Sea

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

Ejemplo 27 *El programa queda implementado en FORTRAN como:*

```
program fdm1d
    implicit none
    integer i, N, nn
    real*8, allocatable :: A(:,:), b(:), x(:)
    real*8 xi, xf, vi, vf, h, R, P, Q, y
    xi = -1.0 ! Inicio del diminio
    xf = 2.0 ! Fin del dominio
    vi = -1.0 ! Valor en la frontera xi
    vf = 1.0 ! Valor en la frontera xf
    nn = 11 ! Particion
    N = nn - 2 ! Nodos interiores
    h = (xf - xi) / (nn-1) ! Incremento en la malla
    print*,"h:",h
    allocate (A(N,N), b(N), x(N))
    R = 1. / (h * h)
    P = -2. / (h * h)
    Q = 1. / (h * h)
    ! Primer renglon de la matriz A y vector b
    A(1,1)=P
    A(2,1)=Q
    call ladoDerecho(xi,y)
    b(1)=y-vi*R
    ! Renglones intermedios de la matriz A y vector b
    do i=2,N-1
        A(i-1,i)=R
        A(i,i)=P
        A(i+1,i)=Q
        call ladoDerecho(xi+h*(i-1),y)
        b(i)= y
    end do
    ! Renglon final de la matriz A y vector b
    A(N-1,N)=R
    A(N,N)=P
    call ladoDerecho(xi+h*N,y)
    b(N)= y -vf*Q
    call gaussSiedel(A, x, b, N, 1000)
     print*, "A: ", A
     print*, "b: ", b
    print*, "x: ", x
end program
subroutine ladoDerecho(x,y)
    real*8, intent(in) :: x
    real*8, intent(inout) :: y
    real*8 pi
    pi = 3.14159265358979323846264338 32;
```

```
        y = -pi * pi * cos(pi * x);
    end subroutine
    subroutine gaussSiedel(a, x, b, nx, iter)
        implicit none
        integer, intent(in) :: nx, iter
        real*8, intent(in) :: a(nx,nx), b(nx)
        real*8, intent(inout) :: x(nx)
        integer i, j, m
        real*8 sum

        do m = 1, iter
            do i = 1, nx
                sum = 0.0
                do j = 1, nx
                    if (i .NE. j) then
                        sum = sum + a(i,j) * x(j)
                    end if
                end do
                x(i) = (1.0 / a(i,i)) * (b(i) - sum)
            end do
        end do
    end subroutine
```

6.8 Implementación en C

C es un lenguaje de programación de tipos de datos estáticos, débilmente tipificado, de medio nivel, existen una gran variedad de subrutinas para manipulación de matrices, pero es facil implementar lo necesario para nuestros fines.

Sea

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

Ejemplo 28 *El programa queda implementado en C como:*

```
#include <math.h>
#include <stdio.h>
#define PARTICION 11 // Tamano de la particion
#define N 9 // Numero de incognitas
#define VISUALIZA 1 // (0) No visualiza la salida, otro valor la visualiza
// Lado derecho de la ecuacion diferencial parcial
double LadoDerecho(double x)
{
    double pi = 3.14159265358979323846264338327;
    return -pi * pi * cos(pi * x);
}
// Resuelve Ax=b usando el metodo Jacobi
void Jacobi(double A[N][N], double x[], double b[], int n, int iter)
```

```
{
    int i, j, m;
    double sum;
    double xt[N];
    for (m = 0; m < iter; m ++)
    {
        for (i = 0; i < n; i++)
        {
            sum = 0.0;
            for (j = 0; j < n; j ++)
            {
                if ( i == j) continue;
                sum += A[i][j] * x[j];
            }
            if (A[i][i] == 0.0) return;
            xt[i] = (1.0 / A[i][i]) * (b[i] - sum);
        }
        for (i = 0; i < n; i++) x[i] = xt[i];
    }
    if (VISUALIZA)
    {
        printf("\nMatriz\n");
        for (i = 0; i < n; i++)
        {
            for (j = 0; j < n; j++)
            {
                printf("%f ", A[i][j]);
            }
            printf("\n");
        }
        printf("\nb\n");
        for (i = 0; i < n; i++)
        {
            printf("%f ", b[i]);
        }
        printf("\nx\n");
        for (i = 0; i < n; i++)
        {
            printf("%f ", x[i]);
        }
        printf("\n");
    }
}
// Resuelve Ax=b usando el metodo Gauus-Siedel
void Gauss_ Siedel(double A[N][N], double x[], double b[], int n, int iter)
{
```

```
        int i, j, m;
        double sum;
        for (m = 0; m < iter; m ++)
        {
            for (i = 0; i < n; i++)
            {
                sum = 0.0;
                for (j = 0; j < n; j ++)
                {
                    if ( i == j) continue;
                    sum += A[i][j] * x[j];
                }
                if (A[i][i] == 0.0) return;
                x[i] = (1.0 / A[i][i]) * (b[i] - sum);
            }
        }
        if (VISUALIZA)
        {
            printf("\nMatriz\n");
            for (i = 0; i < n; i++)
            {
                for (j = 0; j < n; j++)
                {
                    printf("%f ", A[i][j]);
                }
                printf("\n");
            }
            printf("\nb\n");
            for (i = 0; i < n; i++)
            {
                printf("%f ", b[i]);
            }
            printf("\nx\n");
            for (i = 0; i < n; i++)
            {
                printf("%f ", x[i]);
            }
            printf("\n");
        }
    }
    int main()
    {
        double xi = -1.0; // Inicio del diminio
        double xf = 2.0; // Fin del dominio
        double vi = -1.0; // Valor en la frontera xi
        double vf = 1.0; // Valor en la frontera xf
```

```
    int n = PARTICION; // Particion
    double h = (xf - xi) / (n - 1); // Incremento en la malla
    int i;
    double A[N][N]; // Matriz A
    double b[N]; // Vector b
    double x[N]; // Vector x
    double R = 1 / (h * h);
    double P = -2 / (h * h);
    double Q = 1 / (h * h);
    // Primer renglon de la matriz A y vector b
    A[0][0] = P;
    A[0][1] = Q;
    b[0] = LadoDerecho(xi) - vi * 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;
        b[i] = LadoDerecho(xi + h * i);
    }
    // Renglon final de la matriz A y vector b
    A[N - 1][N - 2] = R;
    A[N - 1][N - 1] = P;
    b[N - 1] = LadoDerecho(xi + h * N - 2) - vf * Q;
    // Resuleve el sistema lineal Ax=b
    Gauss_Siedel(A, x, b, N, 1000);
    Jacobi(A, x, b, N, 1000);
    return 0;
}
```

7 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 Bounduary Value Problems and Applicarions* 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 Diferential 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 Aplications*, 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. Skjellem, *Using MPI, Portable Parallel Programming Whit 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. Engng., 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-symetric 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] Agustín Alberto Rosas Medina, *El Número de Péclet y su Significación en la Modelación de Transporte Difusivo de Contaminantes,* Tesis para obtener el titulo de Matemático, UNAM, 2005.

[36] Omar Jonathan Mendoza Bernal, Resolución de Ecuaciones Diferenciales Parciales Mediante el Método de Diferencias Finitas y su Paralelización, Tesis para obtener el titulo de Matemático, UNAM, 2016.

[37] 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.

[38] Tezaur R., *Analysis of Lagrange Multipliers Based Domain Decomposition*. P.H. D. Thesis, University of Colorado, Denver, 1998.

[39] Herrera I. & Rubio E., *Unified Theory of Differential Operators Acting on Discontinuos 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.

[40] 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, Junuary 7-9, 1987.

[41] Toselli, A., *FETI Domain Decomposition Methods for Escalar Advection-Diffusion Problems*. Compututational Methods Appl. Mech. Engrg. 190. (2001), 5759-5776.

[42] C.T. Keller, *Iterative Methods for Linear and Nonlinear Equations*, Societe for Industrial and Applied Mathematics, 1995.

[43] 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.

[44] Zdengk Dostdl 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).

[45] Charbel Farhat, Michel Lesoinne, and Kendall Pierson. *A Scalable Dual-Primal Domain Decomposition Method*. Numer. Linear Algebra Appl., 7(7-8):687-714, 2000.

[46] 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.

[47] 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.

[48] 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.

[49] 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.

[50] 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, Virgina, May 6-8, 1991.

[51] 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.

[52] Zhangxin Chen, Guanren Huan and Yuanle Ma. *Computational Methods for Multiphase Flow in Porous Media*, SIAM, 2006.

[53] Jos Stam. SIGGRAPH 99, Proceedings of the 26th annual conference on Computer Graphics and Interactive Techniques, Pages 121-128, ACM, 1999.