ISSN Electrónico: 2215-3470
DOI: http://dx.doi.org/10.15359/ru.33-1.7

UNICIENCIA Vol. 33, N° 1, pp. 83-100. Enero-Junio, 2019
URL: www.revistas.una.ac.cr/uniciencia
Correo electrónico: revistauniciencia@una.cr

Aspectos computacionales del método de diferencias finitas para la ecuación de calor dependiente del tiempo

Computational aspects of the finite difference method for the time-dependent heat equation

Aspectos computacionais do método de diferenças finitas para a equação de calor dependente do tempo

Filánder Sequeira-Chavarría
filander.sequeira@una.ac.cr
Escuela de Matemática
Universidad Nacional
Heredia, Costa Rica
Orcid: https://orcid.org/0000-0002-0593-3446

Melvin Ramírez-Bogantes
meramirez@itcr.ac.cr
Escuela de Matemática
Instituto Tecnológico de Costa Rica
Cartago, Costa Rica
Orcid: https://orcid.org/0000-0001-5516-0085

Recibido-Received: ***1/abr/2018*** • Corregido-Corrected: ***4/jun/2018.***
Aceptado-Accepted: ***13/ago/2018*** • Publicado-Published: ***31/ene/2019.***

Resumen

En este artículo se describe en detalle un algoritmo para la eficiente implementación computacional del método de diferencias finitas (MDF) en la ecuación de calor dependiente del tiempo, con condiciones de frontera de Dirichlet no homogéneas, en dos dimensiones. Para validar el método presentado aquí se utiliza el paquete computacional MATLAB®, sin embargo, los procesos se exponen independientes al lenguaje de programación. Finalmente se presentan resultados numéricos que validan el algoritmo propuesto.

Palabras claves: ecuación de calor; método de diferencias finitas; implementación computacional; MATLAB®

Abstract

In this paper we describe in detail an algorithm for the efficient computational implementation of the finite difference method (FDM) in the two-dimensional time-dependent heat equation with non-homogeneous Dirichlet boundary conditions. The MATLAB® software was used to validate the method mentioned here; however, the processes are presented independently from the programming language. Finally, numerical results are presented to validate the proposed algorithm.

Keywords: Heat equation; finite difference method; computational implementation; MATLAB®

ISSN Electrónico: 2215-3470
DOI: http://dx.doi.org/10.15359/ru.33-1.7

Resumo

Neste artigo é descrito detalhadamente um algoritmo para a eficiente implementação computacional do método de diferenças finitas (MDF) na equação de calor dependente do tempo, com condições de fronteira de Dirichlet não homogêneas, em duas dimensões. Para validar o método apresentado aqui é utilizado o pacote computacional MATLAB®, porém, os processos são expostos independentes da linguagem de programação. Finalmente, são apresentados os resultados numéricos que validam o algoritmo proposto.

Palavras-chaves: Equação de calor; método de diferenças finitas; implementação computacional; MATLAB®

UNICIENCIA Vol. 33, N°. 1, pp. 83-100. Enero-Junio, 2019 • URL: www.revistas.una.ac.cr/uniciencia • Correo electrónico: revistauniciencia@una.cr

El presente artículo nace como un mecanismo de difusión para los métodos numéricos en la enseñanza superior para carreras de Ingeniería, Computación y Matemática. Su principal contribución no es la teoría matemática y física que envuelve los métodos numéricos para las Ecuaciones Diferenciales Parciales (EDP). Si no más bien, un documento que recopila todos aquellos aspectos importantes sobre una eficiente y clara implementación computacional de un método numérico de aproximación para EDP. Se espera que lo detallado aquí pueda ser utilizado como apoyo en los cursos de postgrado y por investigadores en sus respectivos trabajos. Es importante aclarar que, si bien, lo presentado aquí puede ser hallado en la literatura, al menos por el conocimiento de los autores, no se cuenta con un escrito que clarifique con gran detalle las técnicas a emplear a la hora de aproximar la solución de EDP por medio del método de diferencias finitas. Adicional a esto, no se cuenta con mucha bibliografía en el idioma español, que describa la implementación computacional del método de diferencias finitas.

En relación con la temática principal de este artículo, se resalta que el análisis y diseño de métodos numéricos para EDP ha sido un área de gran investigación en las últimas décadas, debido a que estas surgen de numerosos problemas de interés como la Ingeniería y la Física (ver, por ejemplo, Haberman (1998), Carslow y Jaeger (1959), y sus referencias). A pesar de lo enriquecedor que es la teoría matemática para la resolución de EDP (ver, por ejemplo, Evans (2010)), lo cierto es que esta no es flexible a la hora de cambiar datos en la ecuación diferencial, ni tampoco brinda, en la mayoría de los casos, la solución explícita de estas. Es debido a ello que la aplicación de métodos numéricos para obtener aproximaciones se convierte, no solo en una gran herramienta, sino en general, en la única para poder resolver un problema que involucre EDP. Aunque claro, llevar los problemas de la matemática a la computación, provoca nuevos inconvenientes. En particular, la dificultad de realizar la implementación de los métodos, lo cual será el punto principal de discusión en este texto.

Para establecer una acotación de la zona en la que se enfoca este trabajo, se debe tener en cuenta que la misma corresponde al uso del Método de Diferencias Finitas (MDF) para una EDP parabólica, conocida como la *ecuación de calor*, dado que esta modela el flujo de temperatura en una región que está bajo la influencia de una fuente de calor. Esa delimitación es de suma importancia, no solo por la variedad

ISSN Electrónico: 2215-3470
DOI: http://dx.doi.org/10.15359/ru.33-1.7

de EDP que se pueden abordar, sino por la cantidad de métodos numéricos existentes para una misma EDP. Para dar un ejemplo de esto, además del enfoque por MDF, el cual consiste en reemplazar cada operador diferencial dentro de la EDP por una estimación puntual del mismo (ver, Ciarlet (1995), LeVeque (2007), Morton y Mayers (2005), Strikwerda (2004), Thomas (1995), y sus referencias); también es posible obtener una aproximación al reemplazar el espacio vectorial donde se ubica la solución exacta por uno de dimensión finita, tal y como ocurre con los Métodos de Elementos Finitos (MEF) (ver Ciarlet (2002)). La diferencia más visual que se presenta entre los MDF y los MEF consiste en que los MEF pueden aproximar la solución de la EDP en dominio con geometrías más complejas que los MDF. A pesar de ello, cuando la EDP depende del tiempo (lo que es usual en los problemas de aplicación), independientemente de si se utiliza MDF o MEF para las variables espaciales, la dependencia del tiempo se discretiza por medio los MDF, dado que los MEF requieren de una relación entre puntos vecinos que resulta inconveniente para la derivada temporal. Esto se puede apreciar en investigaciones recientes como, por ejemplo, en (Guzmán, Shu y Sequeira, 2017).

Por otro lado, basta realizar una búsqueda rápida en MathSciNet para notar que muchas de las investigaciones actuales, que involucran resolver EDP, utilizan el paquete computacional MATLAB® para sus implementaciones. La principal razón de esto obedece a la potencia y facilidad que brinda este programa a la hora de emplear métodos numéricos en un determinado problema. Sin embargo, ningún programa computacional cuenta con todos los métodos numéricos que existen, sobre todo los nuevos, por lo que se hace necesario que los interesados en resolver una EDP deban programar, lo cual no es una tarea fácil, en la mayoría de los casos en. Por tal razón, en este artículo se discuten los aspectos más relevantes a la hora de realizar una eficiente implementación computacional. Estos aspectos se exponen en general para cualquier lenguaje de programación, pero se ejemplificarán con ayuda de MATLAB®.

Finalmente, respecto a la organización de este artículo, se inicia con el problema modelo a desarrollar, el cual corresponde a la ecuación de calor en dos dimensiones con condiciones de Dirichlet no homogéneas y cuya solución depende del tiempo. El dominio de definición de la ecuación será un rectángulo por lo que la aplicación del método de diferencias finitas es idónea en este caso. Luego, se detalla cómo la aproximación por esquemas de diferencias finitas en los operadores diferenciales permite obtener un sistema de ecuaciones lineales, cuya solución corresponde a la evaluación de esta buscada en ciertos puntos particulares. Se precisa, además, la construcción (o ensamblaje) eficiente de este sistema lineal para poder implementar la misma en cualquier lenguaje de programación. En particular se presenta ese ensamblaje con ayuda de MATLAB®. Seguidamente, se consideran aspectos sobre los métodos a utilizar para resolver los sistemas lineales que se generan en cada paso de tiempo. Al final, se muestran algunos experimentos numéricos para validar las programaciones propuestas. Es decir, se comprueba el funcionamiento correcto del código generado. Más aún, se anexa un procedimiento, por medio de videos, para visualizar el comportamiento de la solución obtenida.

UNICIENCIA Vol. 33, N°. 1, pp. 83-100. Enero-Junio, 2019 • URL: www.revistas.una.ac.cr/uniciencia • Correo electrónico: revistauniciencia@una.cr

ISSN Electrónico: 2215-3470
DOI: http://dx.doi.org/10.15359/ru.33-1.7

Método de diferencias finitas para la ecuación de calor

Por simplicidad en la descripción del modelo a trabajar, defina Ω como el cuadrado unitario en IR^2, definido como:

$$\Omega := [0,1] \times [0,1] = \{(x,y) \in IR^2 / 0 \leq x \leq 1, 0 \leq y \leq 1\}$$

Asimismo, sobre Ω se considera la siguiente ecuación parabólica con condiciones de Dirichlet no homogéneas, conocida como la *ecuación de calor*:

$$\begin{cases} u_t = \Delta u + f, & \text{para } (x, y) \in \Omega, t \in]0, T], \\ u(t, x, y) = g(t, x, y), & \text{para } (x, y) \in \partial\Omega, t \in [0, T] \\ u(0, x, y) = u_0(x, y), & \text{para } (x, y) \in \Omega, \end{cases} \quad (1)$$

donde $u_t := \frac{\partial u}{\partial t}$ corresponde a la derivada temporal, $\Delta u := u_{xx} + u_{yy} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$ es el operador del laplaciano, $f = f(t, x, y)$ el término fuente y $T > 0$. El objetivo de este escrito es hallar una aproximación para la solución del sistema (1), para lo cual se emplea el método de diferencias finitas (ver, por ejemplo, Morton y Mayers (2005), Strikwerda (2004) o LeVeque (2007)). Más precisamente, dado un entero $M > 0$, se considera una partición uniforme del intervalo [0, T], dada por:

$$t_n := nk, \text{ para todo } \quad n = 0,1,\ldots, M,$$

donde $k = \Delta t = \frac{T}{M}$, se conoce como el paso en tiempo. Seguidamente, dado un entero $N > 0$, se construye una malla cartesiana (o cuadrícula) uniforme para Ω, la cual consiste de $(N + 2)^2$ puntos (x_i, y_j), donde $x_i := ih$ y $y_j := jh$, con $i, j = 0, 1,\ldots, N+1$ y $h = \frac{1}{N+1}$ es el paso en espacio. Es importante aclarar que esta descomposición de Ω no tiene por qué se ser uniforme, sin embargo, se sigue de esta forma con el único fin de simplificar los cálculos posteriores.

Por otro lado, considere el punto (t_n, x_i, y_j) el cual al ser evaluado en la ecuación diferencial se obtiene que:

$$u_t(t_n,x_i,y_j) = u_{xx}(t_n,x_i,y_j) + u_{yy}(t_n, x_i, y_j) + f(t_n,x_i,y_j),$$

donde, al definir las aproximaciones:

$$U_{ij}^n \approx u(t_n, x_i, y_j) \text{ y } U_{ij}^n \approx u_t(t_n, x_i, y_j),$$

así como $f_{ij}^n := f(t_n, x_i, y_j)$, en conjunto con *el esquema de 5 puntos de esténcil* para el laplaciano (ver, por ejemplo, Morton y Mayers (2005), Strikwerda (2004) o LeVeque (2007)):

$$\Delta u(t_n, x_i, y_j) \approx \frac{1}{h^2}(U_{i-1,j}^n + U_{i,j-1}^n - 4U_{ij}^n + U_{i,j+1}^n + U_{i+1,j}^n), \quad (2)$$

se concluye que

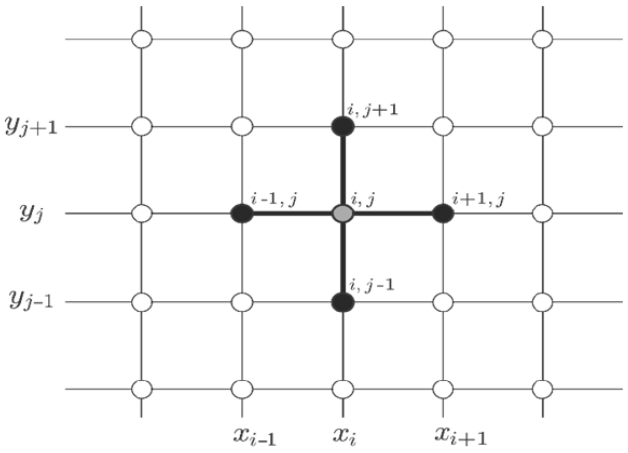
$$U_{ij}^n = \frac{1}{h^2}(U_{i-1,j}^n + U_{i,j-1}^n - 4U_{ij}^n + U_{i,j+1}^n + U_{i+1,j}^n) + f_{ij}^n \quad (3)$$

Con respecto al espacio, este esquema relaciona a cinco puntos de la malla para estimar $\Delta u(t_n, x_i, y_j)$ tal y como se muestra en la Figura 1. Mientras que, con respecto al tiempo, se tiene una ecuación diferencial ordinaria, cuya solución será aproximada por medio del *método de Euler implícito* (ver, por ejemplo, Strikwerda (2004)). La razón de emplear un esquema implícito es para generar un método que sea *incondicionalmente estable* cuando se avanza en cada paso de tiempo. Es decir, no hay condición CFL (Courant-Friedrichs-Lewy) en este caso.

UNICIENCIA Vol. 33, N°. 1, pp. 83-100. Enero-Junio, 2019 • URL: www.revistas.una.ac.cr/uniciencia • Correo electrónico: revistauniciencia@una.cr

ISSN Electrónico: 2215-3470
DOI: http://dx.doi.org/10.15359/ru.33-1.7

Figura 1. Los cinco puntos involucrados en la aproximación del laplaciano.



Nota: Fuente propia de la investigación.

El siguiente paso consiste en emplear el *método de Euler implícito* (conocido también como "backward Euler") para la derivada temporal. En efecto, aproxima el valor de la derivada respecto del tiempo, mediante el esquema de diferencias finitas:

$$U_{ij}^{n} \approx \frac{U_{ij}^{n} - U_{ij}^{n-1}}{k},$$

donde, al sustituir este en (3), se obtiene el esquema de aproximación:

$$\frac{U_{ij}^{n} - U_{ij}^{n-1}}{k} = \frac{1}{h^2}\left(U_{i-1,j}^{n} + U_{i,j-1}^{n} - 4U_{ij}^{n} + U_{i,j+1}^{n} + U_{i+1,j}^{n}\right) + f_{ij}^{n},$$

o bien,

$$U_{ij}^{n} = \frac{k}{h^2}\left(U_{i-1,j}^{n} + U_{i,j-1}^{n} - 4U_{ij}^{n} + U_{i+1,j}^{n}\right) + kf_{ij}^{n} + U_{ij}^{n-1}.$$

Luego, despejando a la derecha los términos de la forma: U^n (es decir, los de potencia n), se reescribe la igual previa de la forma:

$$-U_{i-1,j}^{n} - U_{i,j-1}^{n} + \left(4 + \frac{h^2}{k}\right) U_{ij}^{n} - U_{i,j+1}^{n} - U_{i+1,j}^{n} = h^2 f_{ij}^{n} + \frac{h^2}{k}\, U_{ij}^{n-1}.$$

Por lo tanto, denotando $\mu := \frac{h^2}{k}$, se cumple que:

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

para todo $i, j = 1, 2, \ldots, N$ y para todo $n = 1, 2, \ldots, M$. Es decir, la ecuación (4) establece una relación entre los valores de la solución de la ecuación de calor en dos aproximaciones de tiempo consecutivos. Por ende, la resolución del problema (1) requiere primero aproximar los valores de u en el primer tiempo t_0 y luego emplear el resultado obtenido para alcanzar los valores en el tiempo siguiente. Sin embargo, nótese de la tercera ecuación de (1) que los valores de u en el primer tiempo t_0 son conocidos *a priori* de manera exacta, es por ello, que a la ecuación $u(0, x, y) = u_0(x, y)$ se le conoce como la *condición inicial*. Más aún, siguiendo la notación previa, esta condición se puede reescribir de la forma:

$$U_{ij}^{0} = u_0(x_i, y_j), \quad \text{para } i, j = 0, 1, 2, \ldots, N, N+1. \quad (5)$$

Por último, de la *condición de borde* (segunda ecuación en (1)), se sabe que $u(t, x, y) = g(t, x, y)$, con g alguna función conocida y continua en la frontera de Ω. Así, empleando nuevamente la notación previa se tiene que:

$$U_{ij}^{n} = g(t_n, x_i, y_j) =: g_{ij}^{n}, \quad \text{para } i, j \in \{0, N+1\}, \; t = 0, 1, \ldots, M. \quad (6)$$

Esquema de diferencias finitas

De la aproximación por diferencias finitas (4), en conjunto con la condición inicial (5) y la condición de borde (6), es posible apreciar que los puntos donde se conoce

UNICIENCIA Vol. 33, N°. 1, pp. 83-100. Enero-Junio, 2019 • URL: www.revistas.una.ac.cr/uniciencia • Correo electrónico: revistauniciencia@una.cr

ISSN Electrónico: 2215-3470
DOI: http://dx.doi.org/10.15359/ru.33-1.7

los valores de u, así como aquellos donde no se conoce (valores a determinar o incógnitas), son los ilustrados en la Figura 2.

Ahora, se debe analizar el esquema (4) para los puntos dados por la condición de borde (6), teniendo en cuenta la relación presentada en la Figura 1. Para ello, nótese que:

- Para $i = 1$, de (4) se obtiene:

$$-U^n_{0,j} - U^n_{1,j-1} + (4+\mu)\,U^n_{1j} - U^n_{1,j+1} - U^n_{2,j} = h^2 f^n_{1j} + \mu\, U^{n-1}_{1j}$$

$$\Rightarrow -g^n_{0,j} - U^n_{i,j-1} + (4+\mu)\,U^n_{1j} - U^n_{1,j+1} - U^n_{2,j} = h^2 f^n_{1j} + \mu U^{n-1}_{1j}$$

$$\Rightarrow U^n_{i,j-1} + (4+\mu)\,U^n_{1j} - U^n_{1,j+1} - U^n_{2,j} = h^2 f^n_{1j} + \mu U^{n-1}_{1j} + g^n_{0,j}$$

- Para $i = N$, de (4) se obtiene:

$$-U^n_{N-1,j} - U^n_{N,j-1} + (4+\mu)\,U^n_{Nj} - U^n_{N,j+1} - U^n_{N+1,j} = h^2 f^n_{Nj} + \mu U^{n-1}_{Nj}$$

$$\Rightarrow -U^n_{N-1,j} - U^n_{N,j-1} + (4+\mu) U^n_{Nj} - U^n_{N,j+1} - g^n_{N+1,j} = h^2 f^n_{Nj} + \mu U^{n-1}_{Nj}$$

$$\Rightarrow -U^n_{N-1,j} - U^n_{N,j-1} + (4+\mu) U^n_{Nj} - U^n_{N,j+1} = h^2 f^n_{Nj} + \mu U^{n-1}_{Nj} + g^n_{N+1,j}$$

- Para $j = 1$, de (4) se obtiene:

$$-U^n_{i-1,1} - U^n_{i,0} + (4+\mu)\,U^n_{i1} - U^n_{i,2} - U^n_{i+1,1} = h^2 f^n_{i1} + \mu U^{n-1}_{i1}$$

$$\Rightarrow -U^n_{i-1,1} - g^n_{i,0} + (4+\mu)\,U^n_{i1} - U^n_{i,2} - U^n_{i+1,1} = h^2 f^n_{i1} + \mu U^{n-1}_{i1}$$

$$\Rightarrow -U^n_{i-1,1} + (4+\mu)\,U^n_{i1} - U^n_{i,2} - U^n_{i+1,1} = h^2 f^n_{i1} + \mu U^{n-1}_{i1} + g^n_{i,0}$$

- Para $j = N$, de (4) se obtiene:

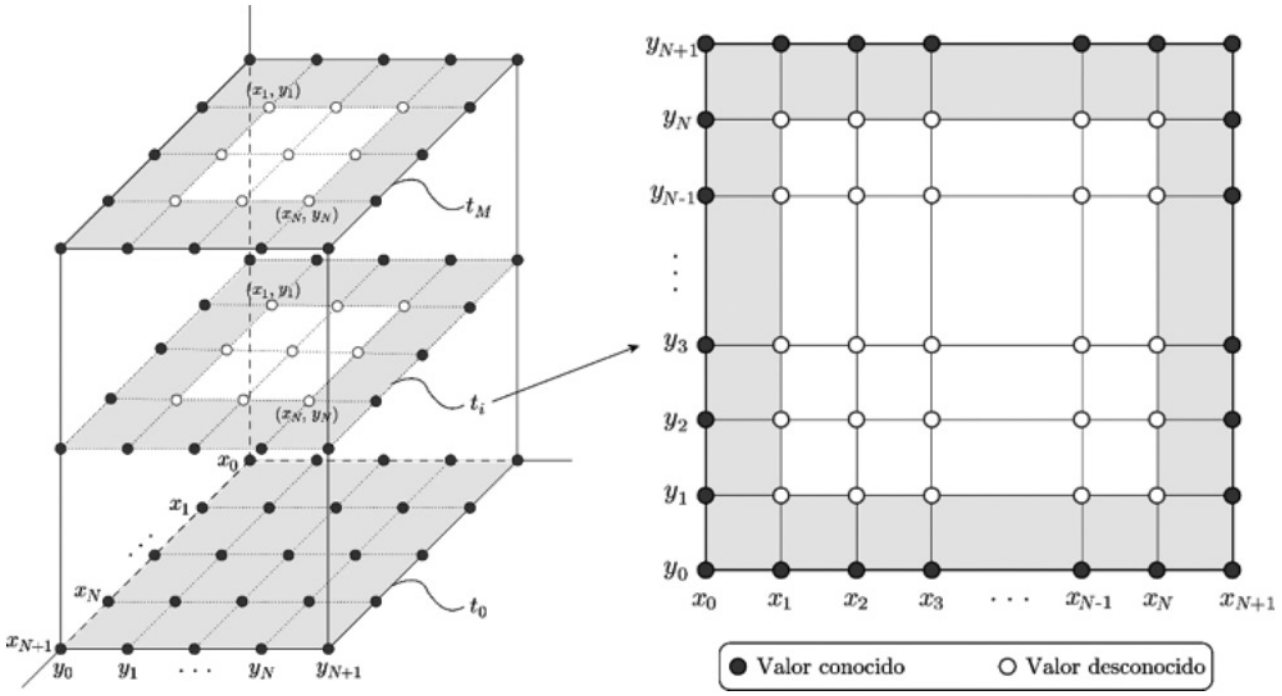
$$-U^n_{i-1,N} - U^n_{i,N-1} + (4+\mu)\,U^n_{iN} - U^n_{i,N+1} - U^n_{i+1,N} = h^2 f^n_{iN} + \mu U^{n-1}_{iN}$$

$$\Rightarrow -U^n_{i-1,N} - U^n_{i,N-1} + (4+\mu) U^n_{iN} - g^n_{i,N+1} - U^n_{i+1,N} = h^2 f^n_{iN} + \mu U^{n-1}_{iN}$$

$$\Rightarrow -U^n_{i-1,N} - U^n_{i,N-1} + (4+\mu)\,U^n_{iN} - U^n_{i+1,N} = h^2 f^n_{iN} + \mu U^{n-1}_{iN} + g^n_{i,N+1}$$

Luego, para los N^2 puntos en los que se desea aproximar la solución (para un tiempo t_n fijo), se pueden ordenar las incógnitas, en el nivel de filas según la Figura 2, en un vector columna en el nivel de bloques, de la forma:

Figura 2. Puntos dados (condición inicial y de borde) e incógnitas por calcular.



Nota: Fuente propia de la investigación.

UNICIENCIA Vol. 33, N° 1, pp. 83-100. Enero-Junio, 2019 • URL: www.revistas.una.ac.cr/uniciencia • Correo electrónico: revistauniciencia@una.cr

ISSN Electrónico: 2215-3470
DOI: http://dx.doi.org/10.15359/ru.33-1.7

$$\boldsymbol{u}_n := \begin{pmatrix} \boldsymbol{u}_n^{[1]} \\ \boldsymbol{u}_n^{[2]} \\ \vdots \\ \boldsymbol{u}_n^{[N]} \end{pmatrix}, \qquad \text{donde} \quad \boldsymbol{u}_n^{[j]} := \begin{pmatrix} U_{1j}^n \\ U_{2j}^n \\ \vdots \\ U_{Nj}^n \end{pmatrix},$$

Para todo $j = 1, 2, \ldots, N$. De esta manera es posible encontrar el vector $\boldsymbol{u}_n \in \mathrm{IR}^{N^2}$, , al resolver un sistema de ecuaciones lineales: $\boldsymbol{A}\boldsymbol{u}_n = \boldsymbol{b}_n$, donde

$$\boldsymbol{A} := \begin{pmatrix} \boldsymbol{T} & -\boldsymbol{I} & \boldsymbol{0} & \cdots & \boldsymbol{0} \\ -\boldsymbol{I} & \boldsymbol{T} & -\boldsymbol{I} & \ddots & \vdots \\ \boldsymbol{0} & -\boldsymbol{I} & \boldsymbol{T} & \ddots & \boldsymbol{0} \\ \vdots & \ddots & \ddots & \ddots & -\boldsymbol{I} \\ \boldsymbol{0} & \vdots & \boldsymbol{0} & -\boldsymbol{I} & \boldsymbol{T} \end{pmatrix} \tag{7}$$

Es una matriz de tamaño $N^2 \times N^2$ con $\boldsymbol{I}$ la matriz identidad de orden N, $\boldsymbol{O}$, la matriz nula de orden N, y $\boldsymbol{T}$ una matriz tridiagonal de orden $\boldsymbol{N}$, definida por

$$\boldsymbol{T} := \begin{pmatrix} 4+\mu & -1 & 0 & \cdots & 0 \\ -1 & 4+\mu & -1 & \ddots & \vdots \\ 0 & -1 & 4+\mu & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & -1 \\ 0 & \vdots & 0 & -1 & 4+\mu \end{pmatrix} \tag{8}$$

Finalmente, el vector de la derecha del sistema viene dado por:

$$\boldsymbol{b}_n := \begin{pmatrix} \boldsymbol{b}_n^{[1]} \\ \boldsymbol{b}_n^{[2]} \\ \vdots \\ \boldsymbol{b}_n^{[N-1]} \\ \boldsymbol{b}_n^{[N]} \end{pmatrix}, \text{ donde} \qquad \boldsymbol{b}_n^{[j]} := \begin{pmatrix} h^2 f_{1j}^n + g_{0j}^n \\ h^2 f_{2j}^n \\ \vdots \\ h^2 f_{N-1,j}^n \\ h^2 f_{Nj}^n + g_{N+1,j}^n \end{pmatrix} + \mu \boldsymbol{u}_{n-1}^{[j]}, \tag{9}$$

Para todo $j = 2, 3, \ldots, N - 1$. En el caso de $j = 1$ y $j = N$, se tiene, respetivamente, que:

$$\boldsymbol{b}_n^{[j]} := \begin{pmatrix} h^2 f_{11}^n + g_{01}^n + g_{10}^n \\ h^2 f_{21}^n + g_{20}^n \\ \vdots \\ h^2 f_{N-1,1}^n + g_{N-10}^n \\ h^2 f_{N1}^n + g_{N+1,1}^n + g_{N0}^n \end{pmatrix} + \mu \boldsymbol{u}_{n-1}^{[1]},$$

Y

$$\boldsymbol{b}_n^{[N]} := \begin{pmatrix} h^2 f_{1N}^n + g_{0N}^n + g_{1,N+1}^n \\ h^2 f_{2N}^n + g_{2,N+1}^n \\ \vdots \\ h^2 f_{N-1,N}^n + g_{N-1,N+1}^n \\ h^2 f_{NN}^n + g_{N+1,N}^n + g_{N,N+1}^n \end{pmatrix} + \mu \boldsymbol{u}_{n-1}^{[1]}.$$

En resumen, es posible encontrar la solución $\boldsymbol{u}_n$ en cada tiempo t_n, con $n = 1, 2, \ldots, M$, mediante la resolución de un sistema lineal $\boldsymbol{A}\boldsymbol{u}_n = \boldsymbol{b}_n$ de orden N^2. Observe que la matriz $\boldsymbol{A}$ no depende de n, es decir, es independiente del tiempo, lo que implica que en cada paso de tiempo se debe resolver un sistema lineal con la misma matriz de coeficientes. Esta última característica es de gran relevancia a la hora de optar por la eficiencia en la resolución del sistema lineal de interés.

Ensamblaje de la matriz de coeficientes

Tal y como se detalla previamente, la matriz de coeficientes $\boldsymbol{A}$ no depende del tiempo, por lo que esta se puede ensamblar una única vez, *a priori* al recorrido de los pasos de tiempo t_n. De esta forma, basta con ensamblar el vector de la derecha $\boldsymbol{b}_n$ en cada t_n, con n=1, 2,…, M, para luego resolver el sistema lineal. En el caso particular del ensamble de $\boldsymbol{A}$,es importante notar que:

$$\boldsymbol{A} := \begin{pmatrix} \boldsymbol{T} & \boldsymbol{0} & \cdots & \cdots & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{T} & \ddots & & \vdots \\ \vdots & \ddots & \boldsymbol{T} & \ddots & \vdots \\ \vdots & & \ddots & \ddots & \boldsymbol{0} \\ \boldsymbol{0} & \vdots & \cdots & \boldsymbol{0} & \boldsymbol{T} \end{pmatrix} +$$

$$\begin{pmatrix} \boldsymbol{0} & -\boldsymbol{I} & \boldsymbol{0} & \cdots & \boldsymbol{0} \\ -\boldsymbol{I} & \boldsymbol{0} & -\boldsymbol{I} & \ddots & \vdots \\ \vdots & -\boldsymbol{I} & \boldsymbol{0} & \ddots & \boldsymbol{0} \\ \vdots & \ddots & \ddots & \ddots & -\boldsymbol{I} \\ \boldsymbol{0} & \vdots & \boldsymbol{0} & -\boldsymbol{I} & \boldsymbol{0} \end{pmatrix} = (\boldsymbol{I} \otimes \boldsymbol{T}) + (\boldsymbol{S} \otimes \boldsymbol{I}),$$

UNICIENCIA Vol. 33, N° 1, pp. 83-100. Enero-Junio, 2019 • URL: www.revistas.una.ac.cr/uniciencia • Correo electrónico: revistauniciencia@una.cr

ISSN Electrónico: 2215-3470
DOI: http://dx.doi.org/10.15359/ru.33-1.7

donde, se considera la matriz $\boldsymbol{S}$ de orden N, definida por:

$$\boldsymbol{S}: = \begin{pmatrix} 0 & -1 & \cdots & 0 \\ -1 & 0 & \ddots & \vdots \\ 0 & -1 & \ddots & 0 \\ \vdots & \ddots & \ddots & -1 \\ 0 & \cdots & -1 & 0 \end{pmatrix} \tag{10}$$

Aquí, el símbolo $\otimes$ corresponde al producto tensorial entre matrices, definido como:

$$\boldsymbol{B} \otimes \boldsymbol{C} = \begin{pmatrix} b_{11}c_{11} & b_{11}c_{12} & \cdots & b_{11}c_{1q} & \cdots & \cdots & b_{1n}c_{11} & b_{1n}c_{12} & b_{1n}c_{1q} \\ b_{11}c_{21} & b_{11}c_{22} & \cdots & b_{11}c_{2q} & \cdots & \cdots & b_{1n}c_{21} & b_{1n}c_{22} & b_{1n}c_{2q} \\ \vdots & \vdots & \ddots & \vdots & & & \vdots & \vdots & \vdots \\ b_{11}c_{p1} & b_{11}c_{p2} & \cdots & b_{11}c_{pq} & \cdots & \cdots & b_{1n}c_{p1} & b_{1n}c_{p2} & b_{1n}c_{pq} \\ \vdots & \vdots & & \vdots & \ddots & & \vdots & \vdots & \vdots \\ b_{m1}c_{11} & b_{m1}c_{12} & \cdots & b_{m1}c_{1q} & \cdots & \cdots & b_{mn}c_{11} & b_{mn}c_{12} & b_{mn}c_{1q} \\ b_{m1}c_{21} & b_{m1}c_{22} & \cdots & b_{m1}c_{2q} & \cdots & \cdots & b_{mn}c_{21} & b_{mn}c_{22} & b_{mn}c_{2q} \\ \vdots & \vdots & \ddots & \vdots & & & \vdots & \vdots & \vdots \\ b_{m1}c_{p1} & b_{m1}c_{p2} & \cdots & b_{m1}c_{pq} & \cdots & \cdots & b_{mn}c_{p1} & b_{mn}c_{p2} & b_{mn}c_{pq} \end{pmatrix}$$

Para $\boldsymbol{B} \in \mathrm{IR}^{mxn}$ y $\boldsymbol{C} \in \mathrm{IR}^{pxq}$, denominado el *producto de Kronecker*. Gracias a este, la construcción de la matriz $\boldsymbol{A}$ se realiza de manera natural (por bloques), lo que además facilita las implementaciones con el paradigma de programación en paralelo (ver, por ejemplo, Quinn (2003)). Así, para ensamblar $\boldsymbol{A}$ basta con ensamblar la matriz $\boldsymbol{S}$ definida en (10) y junto con la matriz identidad $\boldsymbol{I}$ y la matriz $\boldsymbol{T}$ definida en (8), todas ellas de tamaño $N \times N$, y efectuar la suma de dos productos de Kronecker: $(\boldsymbol{I} \otimes \boldsymbol{T}) + (\boldsymbol{S} \otimes \boldsymbol{I})$.

Si se utiliza MATLAB® para llevar acabo la implementación del método propuesto en este escrito, entonces, dado que este posee estructuras para la manipulación de matrices esparcidas (o ralas), es recomendable almacenar la matriz $\boldsymbol{A}$ en dichas estructuras con el fin de mejorar significativamente la eficiencia del método. En efecto, por medio de los comandos para matrices esparcidas de MATLAB®, es posible ensamblar la matriz $\boldsymbol{A}$ mediante la función:

```
function [ A ] = Laplacian2DMatrix( N,
    mu )
    % Matriz identidad
    I = speye( N );
    % Matriz T
    e = -1 * ones( N, 1 );
    T = spdiags( [e, -(4 + mu)*e, e],
    [-1, 0, 1], N, N );
    % Matriz S
    S = spdiags( [e, e], [-1, 1], N,
    N );
    % Matriz A
    A = kron( I, T ) + kron( S, I );
end
```

Donde la rutina `kron(B,C)` realiza l producto de Kronecker: B $\otimes$ C.

Ensamblaje del vector de la derecha

A diferencia de la matriz $\boldsymbol{A}$, la cual se calcula una sola vez, el vector de la derecha $\boldsymbol{b}_n$, definido en (9), debe ser ensamblado en cada paso de tiempo t_n, con $n = 1,2,\ldots, M$. Más aún, este depende del término fuente $f_{ij}^n := f(t_n, x_i, y_j)$, el dato de frontera $g_{ij}^n := g(t_n, x_i, y_j)$, así como de la solución del tiempo anterior $\boldsymbol{u}_{n-1}$. Es importante recordar aquí que la solución en el primer tiempo t_0, denominada $\boldsymbol{u}_0$, es definida a través de la condición inicial, es decir, $U_{ij}^0 := u_0(x_i, y_j)$.

A diferencia del ensamblaje de $\boldsymbol{A}$, donde se toma ventaja de su estructura e independencia respecto de n, el vector de la derecha $\boldsymbol{b}_n$ se ensambla exactamente como fue definido en (9). Más precisamente, definiendo en MATLAB® un tiempo `tn`$= t_n$ fijo, con $n = 1, 2, \ldots, M$, así como la solución previa `u` $= \boldsymbol{u}_{n-1} \in \mathrm{IR}^{N^2}$, se puede determinar el vector `b` $= \boldsymbol{b}_n \in \mathrm{IR}^{N^2}$ por medio de la siguiente función:

ISSN Electrónico: 2215-3470
DOI: http://dx.doi.org/10.15359/ru.33-1.7

```
function [ b ] = Lapacian2DRhs( N, tn,
      x, y, h, f, g, mu, u )
      b = zeros(N^2,1);
      k = N^2 - N;
      v = y(1);     % para j = 1
      w = y(N);     % para j = N
      for i = 1 : N
        z = x(i);
        b(i) = h^2 * f(tn, z, v) +
      g(tn, z, 0);
        b(k + i) = h^2 * f(tn, z, w) +
      g(tn, z, 1);
      end
      b(1)   = b(1)   + g(tn, 0, v);
      b(N)   = b(N)   + g(tn, 1, v);
      b(k+1) = b(k+1) + g(tn, 0, w);
      b(k+N) = b(k+N) + g(tn, 1, w);
      k = N;   % para j = 2, 3 ,..., N
      -1
      for j = 2 : N-1
      v = y(j);
        for i = 1 : N
           b(k + i) = h^2 * f(tn, x(i),
      v);
        end
        b(k+1) = b(k+1) + g(tn, 0, v);
        k = k + N;
        b(k) = b(k) + g(tn, 1, v);
      end
b = b + mu * u;      % contribución de
      la solución anterior
end
```

donde `x` y `y` representan los vectores que almacenan la partición en el espacio, mientras que las funciones `f` y `g`, son implementaciones respectivas de la fuente f y la información de borde g, las cuales pueden ser implementadas en MATLAB®, respectivamente, de la siguiente manera:

```
function [ z ] = f( t, x, y )
      z = 0; % aquí el criterio de la
      función
return
function [ z ] = g( t, x, y )
      z = 0; % aquí el criterio de la
      función
end
```

Con ayuda del fragmento de código previo es posible ensamblar el vector $\boldsymbol{b}_n$, y, dada la matriz $\boldsymbol{A}$, basta con resolver el sistema lineal $\boldsymbol{A}\boldsymbol{u}_n = \boldsymbol{b}_n$, para obtener una aproximación de la solución del problema (1) en el tiempo $t_n = nk$.

Evolución en el tiempo

Ahora que se conoce cómo ensamblar el sistema lineal para cada $n = 1, 2,\ldots, M,$, se debe generar el algoritmo principal que recorre cada tiempo t_n, desde $n = 1$ a $n = M$, obteniendo estos sistemas, para luego proceder a resolverlos, por medio de un método directo o uno iterativo. Para iniciar, considere las descomposiciones del espacio y el tiempo, respetivamente, de la forma:

```
% Malla uniforme para [0,1]x[0,1]
h = 1 / ( N + 1 );
x = h : h : ( 1 - h );
y = x;
% Malla uniforme para [0,T]
k = T / M;
t = k : k : T;
```

donde, se ignoran $x_0 = y_0 = 0$ $x_{N+1} = y_{N+1} = 1$ y $t_0 = 0$, debido a que son valores para los cuales la solución es completamente conocida, según la Figura 2. Por otro lado, es necesario definir la solución inicial `u`=$\boldsymbol{u}_0$ para el tiempo $t_0 = 0$ la cual es dada por la condición inicial $u_0(x, y)$, tal y como se muestra en el siguiente fragmento de código:

```
u = zeros( N^2, 1 );
for j = 1 : N
     for i = 1 : N
       u( (j-1)*N + i ) = u0( x(i),
     y(j) );
     end
end
```

donde la rutina `u0` se puede implementar de la forma:

UNICIENCIA Vol. 33, N°. 1, pp. 83-100. Enero-Junio, 2019 • URL: www.revistas.una.ac.cr/uniciencia • Correo electrónico: revistauniciencia@una.cr

ISSN Electrónico: 2215-3470
DOI: http://dx.doi.org/10.15359/ru.33-1.7

```
function [ z ] = u0( x, y )
      z = 0; % aquí el criterio de la
      función
end
```

Finalmente, basta con recorrer cada paso en tiempo t_n, n = 1, 2,…M, ensamblando el sistema lineal $\boldsymbol{A}\boldsymbol{u}_n = \boldsymbol{b}_n$, y como se realiza en la siguiente rutina de MATLAB®:

```
% Discretización de la ecuación de
      calor
%     u_t = u_xx + u_yy + f, (x,y) en
      [0,1]x[0,1]
%     u = g(t, x, y), (x,y) en la
      frontera
%     u = u_0(x, y), para t = 0
% donde t pertenece a [0,T] y N es el
      número de puntos en
% espacio, mientras que M es el número
      de puntos en tiempo.
function [u] = HeatEquation2DScheme(
      T, N, M, f, g, u0 )
      % Malla uniforme para [0,1]x[0,1]
      h = 1 / (N + 1);
      x = h : h : (1 - h);
      y = x;
      % Malla uniforme para [0,T]
      k = T / M;
      t = k : k : T;
      % Solución en el tiempo t = 0
      u = zeros(N^2, 1);
      for j = 1 : N
        for i = 1 : N
          u( (j-1)*N + i ) = u0( x(i),
      y(j) );
        end
      end
      % Obtener la matriz
      mu = h^2 / k;
      A = Laplacian2DMatrix( N, mu );
      % Recorrer cada paso de tiempo
      for n = 1 : M
        b = Lapacian2DRhs( N, t(n), x,
      y, h, f, g, mu, u );
        u = A \ b; % Resolver el
      sistema lineal
      end
end
```

Observación:

Para validar este fragmento de código, basta con dar una función $u\ (t, x, y)$ y determinar la función $f\ (t, x, y)$ para la cual u es solución del problema (1). Para luego verificar que la solución obtenida por el algoritmo anterior concuerda (bajo alguna medida del error) con la solución exacta u dada inicialmente. Por ejemplo, entre varias de las funciones u con las que se validó este algoritmo se cuenta con:

$$u\ (t, x, y) : = \text{sen}\ (t + \pi\, x)\ \text{sen}\ (t + \pi\, y),$$

con la cual:

$$f\ (t, x, y) : = \cos\ (t + \pi x)\ \text{sen}\ (t + \pi y) + \text{sen}\ (t + \pi x)\ (\cos\ (t + \pi y) + 2\pi^2\ \text{sen}\ (t + \pi y)),$$

y claramente, $g\ (t, x, y) = u\ (t, x, y)$, mientras que $u_0\ (x, y) = u\ (0, x, y)$. Los resultados se pueden apreciar más adelante en la sección de experimentos numéricos.

Resolución del sistema lineal

Según el análisis de las secciones previas, con el fin de aproximar la solución del problema (1), se debe resolver el sistema lineal $\boldsymbol{A}\boldsymbol{u}_n = \boldsymbol{b}_n$, en cada paso de tiempo t_n , donde $\boldsymbol{A} \in \mathbb{R}^{N^2 \times N^2}$ está definida en (7), mientras que $\boldsymbol{b} \in \mathbb{R}^{N^2}$ está definido en (9). Luego de obtener dicho sistema, es necesario resolverlo por algún método directo o iterativo. En el fragmento de código anterior se emplea la instrucción `u = A \ b`, correspondiente al método de *Eliminación Gaussiana*, el cual es un método directo poco estable y poco eficiente. Aunque cabe mencionar que la implementación de MATLAB® es de las más eficientes que existen, dado que además realiza los cálculos en paralelo (ver, por ejemplo, Hahn y Valentine (2016)).

UNICIENCIA Vol. 33, N°. 1, pp. 83-100. Enero-Junio, 2019 • URL: www.revistas.una.ac.cr/uniciencia • Correo electrónico: revistauniciencia@una.cr

ISSN Electrónico: 2215-3470
DOI: http://dx.doi.org/10.15359/ru.33-1.7

A continuación, se detallan brevemente algunas opciones eficientes para resolver los sistemas lineales en cada paso de tiempo, tomando provecho principal de la estructura particular de la matriz de coeficientes y del hecho de que esta es invariante en cada paso. Para mayor información sobre los métodos de resolución enunciados aquí, se puede consultar literatura relacionada con álgebra lineal numérica, como, por ejemplo: Datta (2010), Demmel (1997) y Saad (2003).

Métodos directos

Además del método de *Eliminación Gaussiana*, es posible aplicar dos variantes adicionales, con el fin de acelerar la resolución de los sistemas lineales involucrados en el algoritmo previo. La primera variante se obtiene al observar que el sistema lineal $\boldsymbol{Au}_n = \boldsymbol{b}_n$ tiene la particularidad de que la matriz $\boldsymbol{A}$ es siempre la misma, para todo n=1,…, M. Con ello, si se calcula inicialmente la *factorización LU* de la matriz $\boldsymbol{A}$ y se usa esta en la solución del sistema, es posible reducir el tiempo de ejecución significativamente. Por otro lado, la segunda variante, consiste en notar que la matriz $\boldsymbol{A}$ es simétrica definida positiva, por lo que en lugar de la *factorización LU*, se puede utilizar la *factorización de Cholesky*, la cual tiene como ventaja adicional la reducción del almacenamiento, dado que ahora se almacena en memoria solo una matriz triangular en lugar de dos.

En el caso de la *factorización LU*, esta puede ser calculada en MATLAB® a través de la instrucción:

```
[L, U] = lu( A );
```

donde `L` es una matriz triangular inferior con unos en su diagonal y `U` es una matriz triangular superior, tales que `A = LU`. De esta forma, el sistema lineal `Au = b`, puede ser resuelto mediante el comando:

```
u = U \ ( L \ b );
```

Luego, de manera similar, la instrucción de MATLAB®:

```
R = chol( A );
```

Retorna una matriz triangular superior `R`, la cual satisface que `A = R'R`. Es decir, determina la *factorización de Cholesky*, donde `R:=L'`. Luego de esto, se puede proceder de manera análoga a la de la *factorización LU*.

Métodos iterativos

Los métodos iterativos para resolver sistemas lineales, aproximan la solución de un sistema $\boldsymbol{Ax}=\boldsymbol{b}$, mediante el cálculo de algunos términos de una sucesión que converge a la solución de este. En relación con los métodos directos, los cuales obtienen la solución en una sola iteración, los iterativos son útiles para resolver sistemas que involucran un gran número de variables (del orden de miles), dado que estos no conllevan un alto costo computacional como los directos. Para poder iniciar un método iterativo, se requiere de una aproximación inicial $\mathbf{x}_0$, usualmente definida como el vector nulo, y entonces se calcula un conjunto de aproximaciones sucesivas $\boldsymbol{x}_k$, para $k = 1, 2, \ldots$,NúmeroMáximoIteraciones.

Dado que la matriz de coeficientes en el método de diferencias finitas es simétrica

UNICIENCIA Vol. 33, N° 1, pp. 83-100. Enero-Junio, 2019 • URL: www.revistas.una.ac.cr/uniciencia • Correo electrónico: revistauniciencia@una.cr

ISSN Electrónico: 2215-3470
DOI: http://dx.doi.org/10.15359/ru.33-1.7

definida positiva, entonces es posible emplear métodos iterativos clásicos como: *Jacobi*, *Gauss-Seidel* y *SOR*. En particular, el método de sobre-relajación o método SOR (Succesive Over Relaxation) corresponde en una aceleración del método de Gauss-Seidel en el que, para definir la iteración, se considera $w \in$ IE, $w \neq 0$, para el cual se resuelve el sistema equivalente $w\boldsymbol{A}\boldsymbol{x} = w\boldsymbol{b}$. Más precisamente, el método de SOR es un método de punto fijo cuya iteración viene dada por:

$$\boldsymbol{x}_{k+1}=(\boldsymbol{L}+\tfrac{1}{w}\boldsymbol{D})^{-1}[\boldsymbol{b}+\left(\left(\tfrac{1-w}{w}\right)\boldsymbol{D}-\boldsymbol{U}\right)\boldsymbol{x}_k],$$

donde $\boldsymbol{L}$ es la parte triangular inferior de $\boldsymbol{A}$, $\boldsymbol{U}$ la parte triangular superior y $\boldsymbol{D}$ la diagonal de $\boldsymbol{A}$. Es importante tener en cuenta que según el Teorema de Ostrowski-Reich, presente por ejemplo en Saad (2003), la iteración de SOR converge siempre que $0<w<2$. Más aún, la matriz de coeficientes generada por el método de diferencias finitas admite un parámetro $\boldsymbol{w}$ óptimo, denominado $\boldsymbol{w}_{opt}$, el cual viene dado por:

$$w_{opt}:=\frac{2}{1+\sqrt{1-\left[\rho\left(\mathrm{I}-\frac{1}{4+\mu}\boldsymbol{A}\right)\right]^2}}$$

donde $\mu:=\frac{h^2}{k}$ y $\rho\,(\bullet)$ corresponde al radio espectral de una matriz, el cual en este caso satisface que:

$$\rho:=\left(\boldsymbol{I}-\frac{1}{4+\mu}\boldsymbol{A}\right)=\text{máx}\left\{\left|1-\frac{\lambda_{\text{mín}}(\boldsymbol{A})}{4+\mu}\right|,\left|1-\frac{\lambda_{\text{máx}}(\boldsymbol{A})}{4+\mu}\right|\right\},$$

con $\lambda_{\text{mín}}(\boldsymbol{A})$ y $\lambda_{\text{máx}}(\boldsymbol{A})$, el mínimo y máximo valor propio de $\boldsymbol{A}$, respectivamente.

Finalmente, otro método iterativo recomendado para matrices simétricas positivas definidas es el *método de Gradiente Conjugado* (CG, por sus siglas en inglés), que consiste en determinar la solución de un sistema lineal **Ax=b,** mediante el cálculo de una familia de vectores ***A*-ortogonales** y con ella se puede escribir la solución del sistema como una combinación lineal de esta. En el caso de MATLAB®, este cuenta con comandos predefinidos para ejecutar el método de *Gradiente Conjugado Precondicionado* (PCG, por sus siglas en inglés), de la forma:

```
x = pcg(A, b, tol, iterMax);
```

donde se desea resolver el sistema `Ax=b`, por medio de una tolerancia relativa al primer residuo `tol` > 0, con un máximo de `iterMax` iteraciones. El PCG se conoce como método precondicionado, dado que si se considera la siguiente descomposición para **A**:

$$\mathbf{A}=\mathbf{L}\mathbf{L}^t+\mathbf{R},\ \text{con}\ \mathbf{R}\neq\mathbf{O},$$

Entonces nótese que $\mathbf{A}\approx\mathbf{L}\mathbf{L}^t$ puede ser usado como un precondicionador, donde $\mathbf{L}\mathbf{L}^t$ se conoce como la *factorización incompleta de Cholesky*, la cual no necesariamente existe, pero en el caso de hacerlo, se espera que esta ayude a mejorar el condicionamiento espectral de la matriz del sistema **Ax = b**, al considerar el nuevo sistema equivalente:

$$\mathbf{M}^{-1}\mathbf{A}\mathbf{x}=\mathbf{M}^{-1}\mathbf{b},\ \text{con}\ \mathbf{M}:=\mathbf{L}\mathbf{L}^t.$$

Para obtener la factorización incompleta de *Cholesky* en MATLAB®, se puede utilizar la instrucción:

`L = ichol( A );` (en Windows) o `L = cholinc( A );` (en Unix),

donde, en el caso que `L` pueda ser creada y sea invertible, entonces se puede ejecutar el método de *Gradiente Conjugado Precondicionado* con la factorización incompleta de Cholesky de la siguiente manera:

```
x = pcg(A, b, tol, iterMax, L, L');
```

ISSN Electrónico: 2215-3470
DOI: http://dx.doi.org/10.15359/ru.33-1.7

Comparación: tiempo de ejecución y experimentos numéricos: Validación de la implementación

Memoria requerida

Considere el siguiente problema de valores iniciales y contorno, para la ecuación de calor sobre el dominio $\Omega:=[0,1]^2$:

$$\left\{\begin{array}{rl} u_t=\Delta u, & \text{para } (x,\, y) \in \Omega,\, t \in]0,1] \\ u(t,\, x,\, y) = 0, & \text{para } (x,\, y) \in \partial\Omega,\, t \in [0,1] \\ u(0,\, x,\, y) = xy\,(1-x)(1-y) & \text{para } (x,\, y) \in \Omega, \end{array}\right\}$$

Es decir, se considera el problema modelo (1) con los datos: $f\,(t,\, x,\, y) := 0$, $g\,(t,\, x,\, y) := 0$, $u_0\,(x,\, y) := xy\,(1-x)\,(1-y)$ y $T := 1$.

Ahora, se procede a determinar la memoria requerida, así como el tiempo de ejecución del algoritmo final presentado en la sección anterior. Se considera $N \in \{19,\, 99,\, 499\}$ y $M := N + 1$, así como cada uno de los métodos (directos e iterativos) de resolución de sistemas lineales descritos previamente.

En la Tabla 1, se determina el total de memoria creada por las implementaciones, donde no es posible conocer con exactitud la requerida por *Eliminación Gaussiana* dado que no se puede acceder a la programación MATLAB® de este método. A pesar de ello, se aprecia, tal y como se esperaba, que los métodos iterativos son los que menos memoria requieren. Por ende, cuando la cantidad de incógnitas sea significativamente grande, los resultados en la Tabla 1 sugieren utilizar métodos iterativos, para así no requerir más memoria que la utilizada para almacenar el sistema lineal generado por el método de diferencias finitas. En la Tabla 2, se muestra el tiempo de ejecución que se requiere previo al recorrido de los pasos temporales. Este tiempo considera la construcción de la matriz $\boldsymbol{A}$, así como aquellos procesos que requieren los métodos de resolución, como, por ejemplo, la determinación de factorizaciones. Observe que *Eliminación Gaussiana* fue el que menos tiempo necesitó, dado que solo se requiere ensamblar el sistema lineal para su ejecución. En la Tabla 3, se muestra el tiempo de ejecución exclusivo de la iteración que recorre cada paso de tiempo, donde se resuelven los sistemas lineales. Este es el tiempo más considerable, dado que depende de la cantidad de puntos considerados para la aproximación de la solución. Luego, nótese de estos resultados, que el método más eficiente corresponde a la factorización de *Cholesky*, lo cual es consecuencia de que previo a esta iteración ya se realizaron procedimientos que agilizan la resolución de los sistemas lineales. Más aún, es interesante observar que *Eliminación Gaussiana* y PCG poseen tiempos similares, a pesar de que el primero se realiza en paralelo. Finalmente, en la Tabla 4, se resumen el tiempo total de ejecución, donde nuevamente se aprecia que resolver los sistemas lineales con la factorización de *Cholesky* resulta ser el procedimiento más veloz, al menos, para los tamaños de problemas considerados. A pesar de ello, es importante observar que el método PCG obtuvo un buen rendimiento, lo que sugiere que utilizar este método no solo no requiere gran capacidad memoria comparado con los otros, sino que además es veloz.

Tabla 1. *Memoria requerida en megabytes (MB)*

N	M	Eliminación Gaussiana	Factorización LU	Factorización Cholesky	SOR	PCG
19	20	0.0192	0.1241	0.1241	0.0379	0.0711
99	100	0.5227	15.3299	15.3299	1.0431	2.4909
499	500	13.2944	1909.2407	1909.2407	26.5774	72.7090

Nota: Fuente propia de la investigación.

ISSN Electrónico: 2215-3470
DOI: http://dx.doi.org/10.15359/ru.33-1.7

Tabla 2. *Tiempo de cómputo independiente de la iteración temporal (en min)*

N	M	Eliminación Gaussiana	Factorización LU	Factorización Cholesky	SOR	PCG
19	20	0.00041	0.00008	0.00002	0.00288	0.00002
99	100	0.00050	0.00465	0.00091	0.01066	0.00102
499	500	0.00506	1.97844	0.18130	3.60424	0.10279

Nota: Fuente propia de la investigación.

Tabla 3. *Tiempo de cómputo únicamente de la iteración temporal (en min)*

N	M	Eliminación Gaussiana	Factorización LU	Factorización Cholesky	SOR	PCG
19	20	0.00027	0.00009	0.00008	0.00046	0.00107
99	100	0.03561	0.01298	0.01290	0.09668	0.03907
499	500	6.80456	4.07359	4.04021	34.71655	7.38384

Nota: Fuente propia de la investigación.

Tabla 4. *Tiempo de cómputo total (en min)*

N	M	Eliminación Gaussiana	Factorización LU	Factorización Cholesky	SOR	PCG
19	20	0.00068	0.00017	0.00009	0.00333	0.00109
99	100	0.03611	0.01763	0.01381	0.10734	0.04009
499	500	6.80962	6.05204	4.22151	38.32079	7.48663

Nota: Fuente propia de la investigación.

Visualización del error en la aproximación

Además, del estudio de los recursos involucrados en el método propuesto, también es importante corroborar que se está aproximando la solución del problema (1) correctamente. Para ello, en esta sección se ilustra numéricamente el comportamiento del error:

$$e_{h,n} := \sqrt{\sum_{i,j=1}^{N} |u(t_n, x_i, y_j) - U_{ij}^n|^2},$$

Respecto al tiempo $n = 0, 1, 2, \ldots, M$, siempre que $h := \frac{1}{N+1} \to 0$, cuando $N \to +\infty$. En este contexto, u representa la solución exacta al problema (1). En particular, se considera el problema (1) que queda completamente definido al considerar la solución exacta:

$$u\,(t, x, y) = \text{sen}\,(t + \pi x)\ \text{sen}\,(t + \pi y),$$

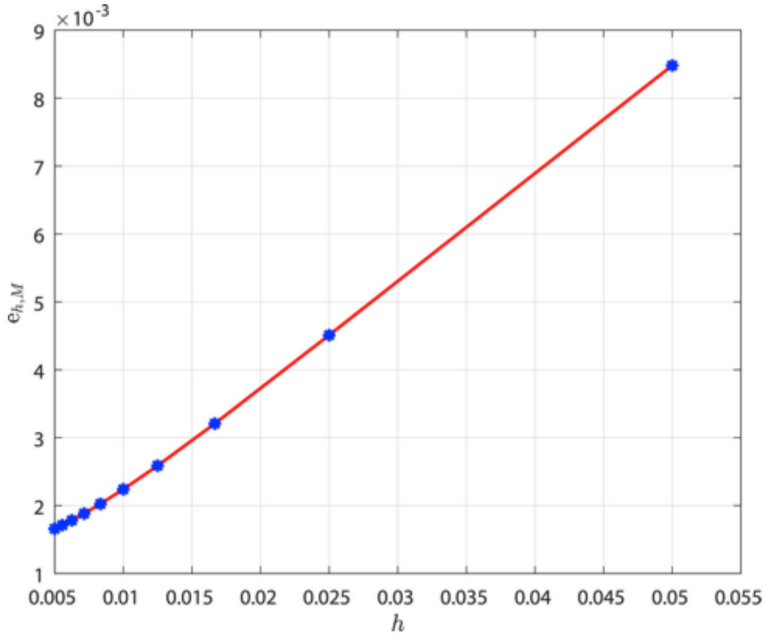
Para $T = 1$. Es decir, se considera $g\,(t, x, y) := u\,(t, x, y)$, $u_0\,(x, y) := u\,(0, x, y)$ y $f\,(t, x, y) := \cos\,(t + \pi x)\ \text{sen}\,(t + \pi y) + (t + \pi x)\,(\cos\,(t + \pi y) + 2\,\pi^2\,\text{sen}\,(t + \pi y))$

Luego, considerando $N \in \{20, 40, 60, \ldots, 200\}$ y $M = 10N$, en la Gráfica 1 se muestra la disminución del error $e_{h,m}$ en el tiempo $t_M = T = 1$. Observe cómo el error decrece a medida que h disminuye.

UNICIENCIA Vol. 33, N°. 1, pp. 83-100. Enero-Junio, 2019 • URL: www.revistas.una.ac.cr/uniciencia • Correo electrónico: revistauniciencia@una.cr

ISSN Electrónico: 2215-3470
DOI: http://dx.doi.org/10.15359/ru.33-1.7

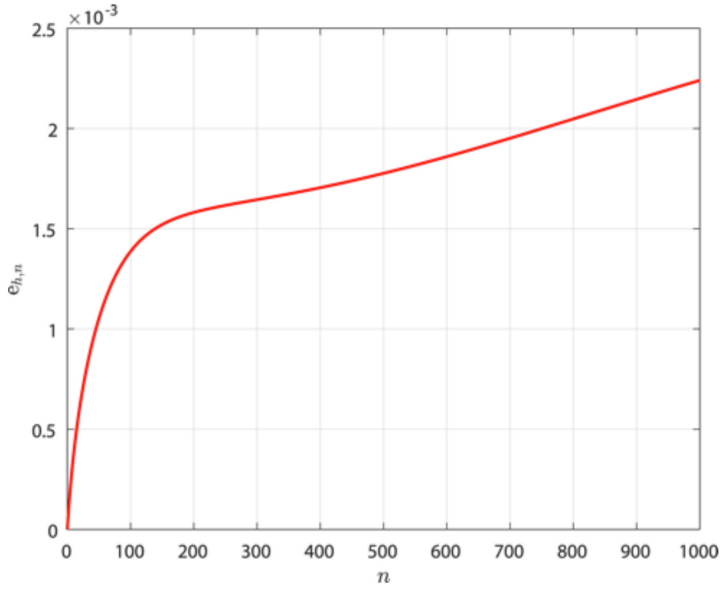
Gráfica 1. *Error* $e_{h,m}$ *para un tiempo fijo*



Nota: fuente propia de la investigación.

Finalmente, considerando $N = 100$ y $M = 1000$, en la Gráfica 2 se muestra el incremento del error $e_{h,n}$, para $n = 0,1,\ldots,M$. Observe cómo el error aumenta según se avanza en el paso en tiempo.

Gráfica 2. Error $e_{h,n}$ para un h fijo.



Nota: fuente propia de la investigación.

Visualización de la solución: Creación de películas en MATLAB®

Debido a que el dominio en el problema (1) está en IR^2 , entonces la solución, respecto al espacio, es una función de tres variables. Sin embargo, la solución depende además del tiempo, por lo que en realidad sería una función de cuatro variables y, por ende, se debe graficar en IR^4, lo cual resulta claro que es visualmente imposible en este escrito, e incluso en la computadora. Por tal razón, se considera realizar una película que muestre el comportamiento de la solución en cada instante t_n, para n=0,1,…,M. Así, el objetivo de esta última sección es mostrar cómo crear una película en MATLAB® que permita visualizar la solución del problema (1). Para ello, considere la siguiente modificación del algoritmo presentado en una sección previa:

```
% Discretización de la ecuación de
     calor
%       u_t = u_xx + u_yy + f,
     (x,y) en [0,1]x[0,1]
%       u = g(t, x, y), (x,y) en la
     frontera
%       u = u_0(x, y), para t = 0
% donde t pertenece a [0,T] y N es
     el número de puntos en
% espacio, mientras que M es el
     número de puntos en tiempo.
function [matU,x,y] =
     HeatEquation2DSchemeMovie(T,
     N, M, f, g, u0)
     % Malla uniforme para [0,1]
     x[0,1]
     h = 1 / (N + 1);
     x = h : h : (1 - h);
     y = x;
     % Malla uniforme para [0,T]
     k = T / M;
```

UNICIENCIA Vol. 33, N°. 1, pp. 83-100. Enero-Junio, 2019 • URL: www.revistas.una.ac.cr/uniciencia • Correo electrónico: revistauniciencia@una.cr

ISSN Electrónico: 2215-3470
DOI: http://dx.doi.org/10.15359/ru.33-1.7

```
      t = k : k : T;
      % Solución en el tiempo t = 0
      u = zeros(N^2, 1);
      for j = 1 : N
        for i = 1 : N
          u( (j-1)*N + i ) = u0( x(i),
      y(j) );
        end
      end
      matU = u; % Matriz donde la
      columna j contiene
        % la solución en el tiempo t_j
      % Obtener la matriz
      mu = h^2 / k;
      A = Laplacian2DMatrix( N, mu );
      % Factorización de Cholesky
      U = chol( A );  L = U';
      % Recorrer cada paso de tiempo
      for n = 1 : M
        b = Lapacian2DRhs( N, t(n), x,
      y, h, f, g, mu, u );
        u = U \ ( L \ b ); % Resolver
      el sistema lineal
        matU = [ matU, u ]; % Almacenar
      la solución
      end
end % Fin del programa
```

Nótese que esta nueva versión realiza las mismas funciones que la anterior, pero dejando el método de *Cholesky* como método de resolución por defecto. Además, retorna la malla en espacio y no la solución en el último tiempo, sino que almacena la solución del tiempo t_j en la columna j de una matriz `matU` de orden $N^2 \times (M+1)$. Esto puede verse como "el guion a ser filmado".

Ahora, considere el siguiente conjunto de instrucciones que permiten realizar la creación de la película:

```
N = 99;
M = 300;
T = 10;
% Determinar la solución aproximada
[matU, x, y] = HeatEquation2DSchemeMovie(
     T, N, M, @f, @g, @u0 );
% Crear película
writerObj = VideoWriter('video.avi');
open( writerObj );
% Control de los ejes coordenados
axis tight
set(gca, 'nextplot', 'replacechildren');
set(gcf, 'Renderer', 'zbuffer');
% Fase de "filmación"
numframes = size(matU, 2);
[X, Y] = meshgrid(x,y);
for n = 1 : numframes
     uh = matU(:, n);
     U = reshape(uh, N, N);
     Z = mesh(X, Y, U);
     title(['Soluci\''on aproximada $t_{',
     num2str(n-1), '}$'],...
       'interpreter','latex')
     xlabel('$x$','interpreter','latex')
     ylabel('$y$','interpreter','latex')
     zlabel('$t$','interpreter','latex')
     currentframe = getframe;
     writeVideo(writerObj, currentframe);
end
close( writerObj );
```

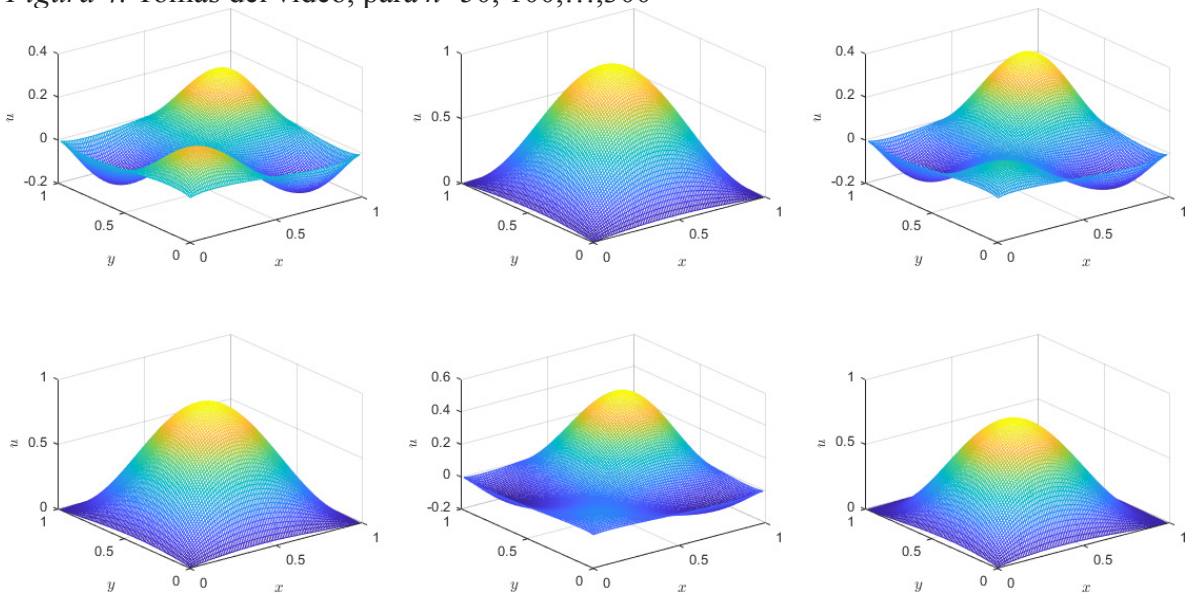
Con ayuda de este fragmento de código, se ejecuta el método y se crea la película para visualizar la solución obtenida. En efecto, considere en el problema (1) los valores: $g(t, x, y) := 0$, $u_0(x, y) := x(1-x)$ y $(1-y)$ y $f(t, x, y) := \cos(t+\pi x)\ \mathrm{sen}(t+\pi x)(\cos(t+\pi y) + 2\,\pi^2 \mathrm{sen}(t+\pi y))$.

En la Figura 4, se presentan algunas imágenes de la película que genera las instrucciones previas.

UNICIENCIA Vol. 33, N°. 1, pp. 83-100. Enero-Junio, 2019 • URL: www.revistas.una.ac.cr/uniciencia • Correo electrónico: revistauniciencia@una.cr

ISSN Electrónico: 2215-3470
DOI: http://dx.doi.org/10.15359/ru.33-1.7

Figura 4. Tomas del video, para n=50, 100,…,300



Nota: Fuente propia de la investigación.

Referencias

Carslow, H. S., y Jaeger, J. C. (1959). *Conduction of Heat in Solids*. Segunda edición. Inglaterra: Oxford University Press.

Ciarlet, P. G. (1995). *Introduction to numerical linear algebra and optimisation*. Estados Unidos: Cambrige University Press.

Ciarlet, P. G. (2002). *The finite Element Method for Elliptic Problems*. Estados Unidos: SIAM. doi: https://doi.org/10.1137/1.9780898719208

Datta, B. N. (2010). *Numerical Linear Algebra and Applications*. Segunda Edición. Estados Unidos: SIAM. doi: https://doi.org/10.1137/1.9780898717655

Demmel, J. W. (1997). *Applied Numerical Linear Algebra*. Estados Unidos: SIAM. doi: https://doi.org/10.1137/1.9781611971446

Evans, L. C. (2010). *Partial Differential Equations*. Segunda edición. Estados Unidos: American Mathematical Society.

Guzmán, J., Shu, C-W., y Sequeira, F. A. (2017). H(div) conforming and DG methods for the incompressible Euler's equations. *IMA Journal of Numerical Analysis*, *37*(4), 1733-1771. doi: https://doi.org/10.1093/imanum/drw054

Haberman, R. (1998). *Elementary Applied Partial Differential Equations, With Fourier Series and Boundary Value Problems*. Tercera edición. Estados Unidos: Prentice-Hall.

Hahn, B. H., y Valentine D. T. (2016). *Essential MATLAB for Engineers and Scientists*. Sexta edición. Estados Unidos: Elsevier.

LeVeque, R. J. (2007). *Finite Difference Methods for Ordinary and Partial Differential Equations, Steady-State and Time-Dependent Problems*. Estados Unidos: SIAM. doi: https://doi.org/10.1137/1.9780898717839

Morton, K. W., y Mayers, D. F. (2005). *Numerical Solution of Partial Differential Equations*. Segunda edición. Inglaterra: Cambrige University Press. doi: https://doi.org/10.1017/CBO9780511812248

Quinn, M. J. (2003). *Parallel Programming in C with MPI and OpenMP*. Estados Unidos: McGraw-Hill.

Saad, Y. (2003). *Iterative Methods for Sparse Linear Systems*. Segunda Edición. Estados Unidos: SIAM. doi: https://doi.org/10.1137/1.9780898718003

Strikwerda, J. C. (2004). *Finite Difference Schemes and Partial Differential Equations*. Segunda edición. Estados Unidos: SIAM. doi: https://doi.org/10.1137/1.9780898717938

UNICIENCIA Vol. 33, N° 1, pp. 83-100. Enero-Junio, 2019 • URL: www.revistas.una.ac.cr/uniciencia • Correo electrónico: revistauniciencia@una.cr

ISSN Electrónico: 2215-3470
DOI: http://dx.doi.org/10.15359/ru.33-1.7

Thomas, J. W. (1995). *Numerical Partial Differential Equations: Finite Difference Methods*. Estados Unidos: Springer-Verlag. doi: https://doi.org/10.1007/978-1-4899-7278-1

UNICIENCIA Vol. 33, N°. 1, pp. 83-100. Enero-Junio, 2019 • URL: www.revistas.una.ac.cr/uniciencia • Correo electrónico: revistauniciencia@una.cr