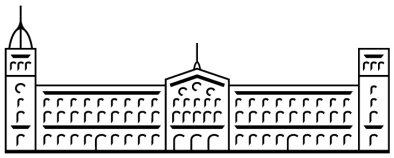


**Treball final de grau**

**GRAU DE MATEMÀTIQUES**

**Facultat de Matemàtiques e Informàtica**
**Universitat de Barcelona**

# Modelización de la ecuación de calor con diferencias finitas

**Autor: Tomás Izquierdo García-Faria**

**Director:** Dr. Àngel Jorba
**Realitzat a:** Departament de Anàlisi i Matemàtica Aplicada

**Barcelona,** 29 de junio de 2017

# Abstract

The aim of this paper is to demonstrate how to use what is learned as a graduate to intuitively resolve a problem and expand the knowledge on it. In this case the heat equation is resolved using some of the numerical methods studied along with a collection of other methods not presented during the math career. The way is solved, using finite differences method means that solving a partial differential equation resumes in using Taylor, resolving an ordinary differential equation and finding the solution of a big dimension sparse linear system. With this process the aim of this project will be satisfied and an approximation of the real temperature for each point in space and time will be found. In order to wite this paper a software was developed where three different ways of resolving differential equations and four different methods to resolve the linear system. This software will output the aproximation using all combinations with the number of iterations, time of execution and the error with the real solution. All together, the reader should have a good understaing of how Finite Difference Method works, some ways to approximate the temperature described with the Dirichlet condition in a given domain and will end up with a first modelling experience.

# Agradecimientos

Me gustaría agradecer a la gente que ha hecho posible que escribiese este trabajo. Lo primero a mi tutor Àngel Jorba, que me ha proporcionado con el suficiente conocimiento, tanto en la carrera como ahora, para elaborar este trabajo. También a mis compañeros de clase que me hayan podido ayudar en las pequeñas dudas que le surgen a uno durante el proceso. Finalmente el apoyo moral que proporciona la familia, como siempre, es importante para todo lo que se hace y en particular para exponer por escrito y oralmente un trabajo de finde grado como este.

# Índice

# 1. Introducción

Los métodos numéricos y el estudio de ecuaciones diferenciales siempre ha sido uno de los campos de la matemática que más me han atraído. El hecho de poder modelar una situación dada, a través de la observación y traducirla a una relación matemática para poder analizarla y sacar conclusión que aporten conocimiento para entenderla mejor, me parece básico para crecer y avanzar como sociedad. Para mí lo más importante era utilizar lo aprendido a lo largo de la carrera para solucionar un problema práctico, con más autonomía de lo que estaba acostumbrado. El problema de aproximar la temperatura de un dominio tridimensional a lo largo del tiempo es bastante atractivo y se puede resolver de forma relativamente sencilla con el método de diferencias finitas. De hecho la solución se basa en conceptos muy utilizados a lo largo de la carrera que resultan fáciles de implementar.

Este trabajo trata de presentar la metodología para afrontar un problema de aproximación de la temperatura en un dominio de dos y tres dimensiones a lo largo de un periodo de tiempo. Para ello se recordarán temas estudiados en la asignatura de métodos numéricos 2 en su mayoría pero teniendo en cuenta otras asignaturas que se tendrán que tener presentes. El texto presentará información de relevancia que hay que conocer y que será crucial para entender el proceso, como teoremas necesarios o la descripción del sistema en el que nos encontramos que vendrá definida por una relación diferencial parcial. Después de explicar el método en sí se procederá a describir cada paso con posibles mejoras a tener en cuenta para finalmente aplicarlo a ejemplos ilustrativos y una visualización de estos para entender el método al completo. Como comentarios y conclusiones se resumen posibles líneas de avance que puede tomar un estudiante aunque existen muchos otros caminos para mejorar el método.

El primer paso es tener clara la pregunta que se quiere resolver y la información que se conoce. Para ello se presentará una de muchas condiciones de entorno donde estará definidas en cierto dominio espacio-temporal, la condición de Dirichlet, donde existen tres condiciones. La primera viene dada por la relación diferencial $u_t = \Delta u + g(x,t)$ con $g(x,t)$ conocida, que se deduce de la ley de Fourier y la ley de transmisión de calor. La segunda condición es conocer el estado del dominio en tiempo inicial, que vendrá denotado por $u_0$. La última condición es la función que definirá la temperatura en la frontera del dominio a lo largo del tiempo. Estas condiciones son suficientes como para encontrar una solución única del sistema y estable en relación a la función de la temperatura en la frontera. Con esto se tendrá un problema sólido para solucionar numéricamente.

Una vez se conocen las condiciones de entorno, en este caso Dirichlet, se empleará el método de diferencias finitas para aproximar la solución. Para ellos se utiliza diferencias divididas para calcular $\Delta u$ que dará lugar a conocer los puntos más cercanos en las tres coordenadas. Con ánimos de expresar estos puntos más cercanos se discretiza el dominio definiendo el número de puntos a utilizar para su representación finita, y así poder hacer referencia a estos puntos más cercanos. Cuando ya se tiene el laplaciano $\Delta u$, quedará un sistema de la forma $u_t = f(u,t)$. Este se resolverá utilizando métodos numéricos para ecuaciones diferenciables ordinarias estudiados

en la universidad como Euler implícito y explícito, notando la inestabilidad del segundo y como afecta esto a la aproximación. También se introducirá el método de Crank-Nicolson que será una aproximación de un orden mayor que los anteriores y permitirá encontrar una aproximación más realista de la solución. Los métodos que si servirán para este contexto serán los implícitos, que darán lugar a un sistema lineal de tantas incógnitas como puntos del interior del haya en el dominio espacial. Para solucionar este sistema se plantean los métodos iterativos porqué la matriz que lo define es una matriz escasa y podemos utilizar la lógica condicional en la programación para calcular el producto de matrices en $n$ pasos (por cada elemento). Esto permitirá no guardar la matriz en memoria y ser más rápidos en los cálculos, mientras que los métodos no-iterativos manipulan las matrices y sí requieren esa memoria. Los métodos más básicos que se presentan, serán también los aprendidos en la carrera Jacobi, Gauss-Seidel y S.O.R (successive over relaxation). Además de estos se presentará el método del gradiente conjugado, que enfoca el problema como un problema de minimización y encuentra la misma aproximación en menos iteraciones y tiempo de ejecución. Todo este proceso se acompañará en cada paso de un ejemplo donde se ve representado todo el proceso en un cuadrado y un cubo en 2 y 3 dimensiones respectivamente. El proceso se ha seguido en otro dominio que me referiré como horno, que consiste en un cuadrado o cubo quitándole un cuadrado o cubo más pequeño de su interior y con altura variable.

Finalmente, cuando ya se tiene una aproximación de la solución para ser visualizada y analizada. Para ello se recogerá la información relevante, como número de iteraciones, tiempo de ejecución y comparación con la solución real. También se guía para representar la solución en dos dimensiones para todo tiempo con ayuda de un video y se presentarán herramientas para visualizar, en tres dimensiones, la solución de las diferentes capas del dominio espacial. Esto puede presentarse como un reto ya que se tienen que representar en una pantalla (2D) cinco dimensiones (temperatura, tiempo y las tres del dominio macizo).

# 2. Información básica

El primer paso para entender una aplicación del método es conocer la información que nos proporciona el propio análisis de la situación. Por ello el siguiente apartado se dedicará a la deducción de la ecuación de calor en tres dimensiones a partir de un análisis físico de la situación. Para llegar a a este análisis se requiere de teoría y por eso se enunciará y demostrará el teorema del máximo, un resultado que va implícito a lo largo del estudio de la ecuación del calor a través del método de diferencias finitas.

## 2.1. Deducción en tres dimensiones

A continuación se resumirá lo estudiado en el grado de matemáticas de la Universidad de Barcelona en la asignatura de modelización, donde se explica como su puede deducir la relación diferencial que ayudará a encontrar una aproximación de la temperatura en cada punto de un dominio discretizado. Esta relación proviene de la ley de Fourier y viene dada por la relación entre el flujo de calor $\bar{q}$ y el gradiente de la temperatura:

$$\bar{q}(X,t) = -K\nabla u(X,t) = -K(\frac{\partial u}{\partial x}(X,t)\overrightarrow{i} + \frac{\partial u}{\partial y}(X,t)\overrightarrow{j} + \frac{\partial u}{\partial z}(X,t)\overrightarrow{k})$$

Suponemos que $u(X,t)$ es la temperatura en un punto $X = (x,y,z)$ en tiempo $t$ con conductividad termina $K > 0$, densidad $\gamma$ y capacidad térmica $c$ constantes y no pasa calor ni escapa al el medio. Entonces, el primer principio de termodinámica dice que el calor $\bar{q}$ que entra por la frontera del dominio espacial $V$, $S = \partial V$, y la energía que se genera en $V$ es igual a la variación de la energía almacenada en $V$. Puede que haya un factor externo que afecte a la temperatura interior que definiremos como $g(X,t)$, si no lo hay $g \equiv 0$.

$$\frac{d}{dt}\int\int\int_V u(X,t)\ dv = -\int\int_S \bar{q}\cdot\vec{n}ds$$

Se integra en un intervalo de tiempo $[0,T]$.

Calor que entra por $S$:

$$\int_0^T \int\int_S \bar{q}\cdot\vec{n}dsdt \overset{\text{T.Gauss}}{=} \overset{\text{Divergencia}}{\int_0^T \int\int\int_V \nabla\cdot\bar{q}\ dvdt} =$$

$$= K\int_0^T \int\int\int_V \delta u(X,t)dvdt \tag{2.1}$$

Energía que se genera en $V$:

$$\int_0^T \int\int\int_V g(X,t)dvdt \tag{2.2}$$

Variación de la energía almacenada en $V$:

$$\int_0^T \int\int\int_V \gamma \cdot c \cdot \frac{\partial u}{\partial t}(X,t)dt \tag{2.3}$$

Utilizando la relación entre las tres integrales (2.1)+(2.2) =(2.3) obtenemos que:

$$\begin{aligned}\int_0^T \int\int\int_V (K\Delta u(X,t) + g(X,t) - \gamma \cdot c\frac{\partial u}{\partial t}(X,t))dv\ dt = 0 \Longrightarrow \\ \Longrightarrow K\Delta u(X,t) + g(X,t) - \gamma c u_t(X,t) = 0 \Longrightarrow \\ \Longrightarrow u_t(X,t) = \frac{K}{\gamma c}\Delta u(X,t) + \frac{g(X,t)}{\gamma c}\end{aligned}$$

Si se define $k^2 := \frac{K}{\gamma c}$ y $f(X,t) := \frac{g(X,t)}{\gamma c}$ se puede conseguir así la definición que resulta más familiar:

$$u_t = k^2\Delta u + f$$

## 2.2. Principio del máximo

A continuación se enunciará el principio del máximo, que nos dice que si una función tiene laplaciano positivo o cero, el máximo estará en la frontera. Esto se utilizará para demostrar, en el caso cuando $u_t = 0$ y por lo tanto $\Delta u = 0$, que la temperatura interior del dominio será menor. También se puede interpretar, cuando un sistema que depende del tiempo está en equilibrio, ya no depende del tiempo. Para la demostración de la unicidad de soluciones de Dirichlet también es útil aplicar el principio del máximo que dice lo siguiente:

**Teorema 2.1** (**Principio del máximo**). *Sea $\Omega$ abierto conexo, regular y compacto en $\mathbb{R}^3$. Supongamos que $u = u(x,y,z)$ es solución de la ecuación laplaciana $\Delta u = 0$ en $\Omega$ y continua en $\bar{\Omega} = \Omega \cup \partial\Omega$. Entonces el máximo y mínimo de $u$ se obtienen en la frontera $\partial\Omega$, esto es:*

$$\exists x_m, x_M \in \partial\Omega \text{ t.q. } u(x_m) \leq u(x) \leq u(x_M)\ \ \forall x \in \Omega$$

*Demostración.* Supongamos que el valor máximo $(x_0,y_0,z_0) \in \Omega$ de la función $u(x,y,z)$ no está en la frontera, por lo tanto la hessiana tendrá la diagonal negativa o cero:

$$u_{xx}(x_0,y_0,z_0) \leq 0,\ \ u_{yy}(x_0,y_0,z_0) \leq 0,\ \ u_{zz}(x_0,y_0,z_0) \leq 0$$

Como $u(x_0,y_0,z_0)$ es un mínimo lo será en particular en las direcciones de eje $x,y,z$ , por lo que se obtiene que

$$u_{xx}(x_0,y_0,z_0) = u_{yy}(x_0,y_0,z_0) = u_{zz}(x_0,y_0,z_0) = 0$$

Definimos la función $v(x,y,z)$ con $\epsilon > 0$ de la siguiente manera:

$$v(x,y,z) = u(x,y,z) + \epsilon(x^2+y^2+z^2)$$

$$\Delta v(x,y,z) = \Delta u(x,y,z) + \epsilon\Delta(x^2+y^2+z^2) = 0 + 6\epsilon$$

Como $\Delta v \leq 0$ a los puntos interiores máximos, esto nos dice que en el interior de $\Omega$ no hay puntos máximos.

Como $u$ es continua en la adherencia de $\Omega$ entonces $v$ también lo és en $\bar{\Omega}$, por lo tanto tiene un máximo $(x_0, y_0, z_0)$ y tiene que estar en $\partial\Omega$

Por construcción de $v$ observamos que para cualquier $(x,y,z) \in \Omega$ tenemos:

$$u(x,y,z) \leq v(x,y,z) \leq v(x_0,y_0,z_0) = u(x_0,y_0,z_0) + \epsilon||(x_0,y_0,z_0)||^2 \leq$$

$$\leq \max_{(x,y,z)\in\partial\Omega} u(x,y,z) + \epsilon \max_{(x,y,z)\in\partial\Omega} ||(x_0,y_0,z_0)||^2 < \infty$$

Si $\epsilon \to 0$ obtenemos:

$$u(x,y,z) \leq \max_{(x,y,z)\in\bar{\Omega}} u(x,y,z) \Longrightarrow \max_{(x,y,z)\in\partial\Omega} u(x,y,z) \leq \max_{(x,y,z)\in\partial\Omega} u(x,y,z)$$

#

## 2.3. Condición de Dirichlet

Una de las cosas más importantes es determinar la información que se tiene a priori del problema, esto son las condiciones de entorno. Hay varias condiciones que pueden alterar el método y cambiar la solución. A continuciacion se presentará la condición de contorno de Dirichlet, que nos proporciona la temperatura en tiempo inicial y sobre las fronteras asociadas al espacio. Se verá como esto es suficiente para encontrar una solución y asegurar que és única. Las condiciones son las siguientes:

Sea $u : \Omega \times [0,1] \subset \mathbb{R}^n \times \mathbb{R} \longrightarrow \mathbb{R}$

$$\begin{cases} u_t = k^2\Delta u + g(u,t) & \\ u(x,0) = u_0(x) & x \in \Omega \\ u(x,t) = f(x,t) & \text{if } x \in \partial\Omega \end{cases} \tag{2.4}$$

con $k \in$ constante y dos funciones $g(u,t) : \mathbb{R} \times [0,1] \longrightarrow \mathbb{R}$ y $f(x,t) : \mathbb{R}^n \times [0,1] \longrightarrow \mathbb{R}$ conocidas.

Estas condiciones determinan una solución única que es la que queremos aproximar utilizando el método de diferencias finitas. A continuación se demostrará la unicidad y estabilidad que proporcionan las condiciones de Dirichlet que son los resultados más básicos. La demostración de la existencia no entra dentro del alcance de este trabajo. No obstante como las funciones $f$ y $g$ de nuestro sistema son diferenciables, donde solo es necesaria la continuidad, se sabe que existe al menos una solución.

El siguiente teorema nos demuestra si existen dos soluciones, tienen que ser la misma:

**Teorema 2.2.** *Sea $u : \Omega \times [0,1] \subset \mathbb{R}^n \times \mathbb{R} \longrightarrow \mathbb{R}$ una función continua como la presentada 2.4, las condiciones de Dirichlet. Entonces, si existe una función $u(x,t)$ solución del sistema, es única.*

*Demostración.* Sea $u_1(x,t), u_2(x,t)$ soluciones de la codición de Dirichlet y tomemos el cambio de variable $w = u_1 - u_2$, entonces nuestro sitema pasa a ser:

$$\begin{cases} w_t - k^2\Delta w = 0 & \\ w(x,0) = 0 & x \in \Omega \\ w(x,t) = 0 & \text{if } x \in \partial\Omega \end{cases} \tag{2.5}$$

Si $\Omega \subset \mathbb{R}^n$ es un abierto acotado no vacío, y $u \in \mathcal{C}^2(\Omega) \cap \mathcal{C}^0(\Omega)$ es una función tal que $-\Delta u \leq 0$ en $\Omega$, entonces:

$$\max_{\bar{\Omega}} u = \max_{\partial\Omega} u$$

En particular, si $u \leq 0$ sobre $\partial\Omega$ entonces $u \leq 0$ en $\bar{\Omega}$

Como $\Omega$ es un abierto acotado no vacío y $w \in \mathcal{C}^2(\Omega) \cap \mathcal{C}^0(\Omega)$ cumple que $-\Delta w \leq 0$ en $\Omega$ utilizando el principio del máximo tenemos que el máximo y el mínimo tienen que estar en la froentera donde $w(x,t) = 0 \ \forall x \in \partial\Omega$, por lo tanto, se obtiene que $w(x,t) \equiv 0$. Finalmente concluimos que las soluciones $u_1(x,t) = u_2(x,t)$ són iguales para todo tiempo $t \geq 0$.

#

Ahora que se sabe de la existencia y unicidad de la solución, se quiere comprobar cuanto cambia la solución, por el cambio de condición frontera.

**Teorema 2.3.** ***Enunciado Estabilidad*** *Sean $u_1, u_2$ soluciones de sistemas que se diferencian únicamente en la condición de frontera $f_1, f_2$ respectivamente, definidas en el dominio $\Omega$*

$$\begin{cases} u_{1t} = k^2\Delta u_1 + g_1(x,t) & \\ u_1(x,0) = u_{1_0} & x \in \Omega \\ u_1(x,t) = f_1(x,t) & si\ x \in \partial\Omega \end{cases} \qquad \begin{cases} u_{2t} = k^2\Delta u_2 + g_2(x,t) & \\ u_2(x,0) = u_{2_0} & x \in \Omega \\ u_2(x,t) = f_2(x,t) & si\ x \in \partial\Omega \end{cases}$$

*entonces se cumple:*

$$\int_\Omega (u_1(x,t) - u_2(x,t))^2 \leq \sum_{i=1}^{n} \int_\Omega (\Phi_1(x_i,t) - \Phi_2(x_i,t))^2$$

*Demostración.* Sea $w = u_1 - u_2$, el del sistema 2.5, si multiplicamos $w$ por la ecuación y utilizando la regla de derivación del producto y la regla de la cadena, obtenemos:

$$0 \cdot w = (w_t - k^2\Delta w) \cdot w = ww_t - wk^2\Delta w = \frac{d}{dt}(\frac{w^2}{2}) + \sum_{i=1}^{n} k \cdot w_{x_i}^2 - \sum_{i=1}^{n} k \cdot (w \cdot w_{x_i})_{x_i}$$

Integrando sobre el dominio $\Omega$ obtenemos:

$$0 = \int_\Omega \frac{d}{dt}(\frac{w^2}{2}) + \int_\Omega \sum_{i=1}^{n} k \cdot w_{x_i}^2 - \int_\Omega k \cdot (w \cdot w_{x_i})_{x_i}$$

$$= \frac{\partial}{\partial t}\int_\Omega \frac{w^2}{2} + k\sum_{i=1}^{n}\int_\Omega w_{x_i}^2 - k\sum_{i=1}^{n}\int_\Omega (w\cdot w_{x_i})_{x_i}$$

Teniendo en cuenta que $\int_\Omega (w\cdot w_{x_i})_{x_i} = \Big|_\Omega w\cdot w_{x_i} = 0$, se tiene:

$$\Rightarrow \frac{\partial}{\partial t}\int_\Omega \frac{w^2}{2} = -k\sum_{i=1}^{n}\int_\Omega w_{x_i}^2 \leq 0 \tag{2.6}$$

Utilizando 2.6 y substituyendo $u_1, u_2$ y las soluciones en estos puntos $\Phi_1(x_i,t), \Phi_2(x_i,t)$ obtenemos:

$$\int_\Omega (u_1(x,t) - u_2(x,t))^2 \leq \sum_{i=1}^{n}\int_\Omega (\Phi_1(x_i,t) - \Phi_2(x_i,t))^2$$

#

Con estas dos demostraciones que demuestran que las soluciones que determinan la condición de Dirichlet existen son únicas y estables, tenemos un problema sólido y con la información suficiente para aproximar esa solución única.

# 3. Método de diferencias finitas para la ecuación de calor

El método de diferencias finitas trata de crear una aproximación de la temperatura para un punto determinado en un tiempo determinado, es decir, para todo el dominio espacio-temporal. Se parte de la relación diferencial $u_t = k^2\Delta u + g(u,t)$ y una serie de condiciones de entorno que acotarán el número de soluciones. Una vez claro el dominio en el que se va estudiar la ecuación de calor, se descretizará. Hecho esto, será posible encontrar una aproximación de $u_t$ y $\Delta u$, a poder ser del mismo orden, para hallar la temperatura en el dominio discretizado dado. La temperatura resultante acaba dependiendo de la de los puntos más próximos, que coincide con la idea intuitiva que se tiene de conducción o del flujo de temperatura en un dominio. Estos puntos serán del mismo tiempo $t$ o $t - \delta t$ dependiendo del método de aproximación de la derivada respecto el tiempo que se utilice (métodos numéricos de resolución de EDOS), que puede ser explícito o implícito. En los casos que se estudiarán, los métodos implícitos, que convergerán a una solución, darán lugar a sistemas lineales que se solucionarán con métodos básicos. La inestabilidad del método explícito llevará a comprobar que a medida que pasa el tiempo el error tiendo a infinito. Como se comprobará, el método de diferencias finitas es un método en su conjunto bastante intuitivo, pero a la vez con resultados positivos en escenarios similares a los que se presentan en este trabajo.

Conociendo las hipótesis del problema de Dirichlet, es decir la temperatura en la frontera y una condición inicial de la temperatura sobre todo el dominio, se propone utilizarlas para encontrar la temperatura para cada punto de un dominio, durante un periodo de tiempo, que viene descrita como la solución de la ecuación diferencial

$$u_t = \kappa^2\Delta u + g(u,t) \tag{3.1}$$

El método de diferencias finitas proprociona una posible solución a este problema aproximando los terminos desconocidos $u_t$ y $\Delta u$ de esta igualdad, a partir de la información que conocemos. Para ello lo primero es discretizar nuestro dominio no temporal de una manera razonable. Los dominios, cerrados y acotados que se representarán como $\Omega$ que se estudiarán, serán la unión de un número $r$ (finito) de cubos $n$-dimensionales de tal forma que

$$\Omega = \bigcup_{i\in\{1,..,r\}} C_i,\ \ C_i = [x_1, y_1] \times ... \times [x_n, y_n] \subset \mathbb{R}^n\ \ x_j, y_j \in \mathbb{N}\ \ x_j < y_j\ \ j \in 1, ..., n$$

será dónde estará definida la ecuación en derivadas parciales (EDP) que necesitamos reemplazar por una ecuación expresada en difrencias finitas (EDF). En los ejemplos prácticos que se ilustrarán nos concentraremos únicamente en los casos $n = 2, 3$, pero el razonamiento es intuitivo para dimensiones más generales.

Se quiere que la EDF esté definida en un número finito de puntos equidistantes y con la densidad suficiente como para perder la menor precisión posible. Este es un proceso que puede parecer muy simple (y lo será en muchos casos) pero se puede complicar dependiendo de la diferenciabilidad del dominio. Para ello se divide la

proyección del dominio sobre cada coordenada (que estará contenida en el intervalo $[0, a_i]$) en $N_i \in \mathbb{N}$ sub-intervalos de una medida dada $\delta x_i << \text{mín}_{j\in\{1,...,n\}} |[x_j, y_j]|$, siguiendo Figura 1.

El dominio espacial finito, que denotaremos como $\Omega_F$, tiene que estar compuesto por puntos equidistantes. Se definine para ello $\delta := \delta x_1 = ... = \delta x_n$, de tal manera que imponemos implícitamente que $N_i = \frac{a_i}{\delta}$, $\forall i$. El dominio espacio-temporal finito $D_F$ en un tiempo $t$ és de la forma $D_F = \{(x_1, ..., x_n, t) \in \Omega_F \times [0, 1]\}\}$ Como $g(u, t)$ y $\kappa$ son conocidos, se puede hacer uso de la fórmula de Taylor encontrar una aproximación $\tilde{\Delta u}$ de $\Delta u$, para encontrarnos con el sistema:

$$u_t \approx \kappa^2 \tilde{\Delta u} + g(u, t) \tag{3.2}$$

Gracias a tener un dominio discretizado tiene sentido plantear hacer referencia al punto más cercano en cierta dirección, que se utilizará para encontrar una expresión, también aproximada, de $\Delta u$.

**Notación:** La temperatura en un punto $(x_1, x_2, ..., x_n, t) \in D_t$ en tiempo $t$, viene representada por la función $u(x_1, ..., x_n, t) =: u_{i_1,...,i_n,t}$ ó $u$, que también anotaremos $u(x_1, ..., x_j + \delta_{x_j}, ..., x_n, t + \delta_t)$ como $u_{i_1,...,i_j+1,...,i_n,t+1}$ ó $u_{i_j+\delta,t+\delta}$. En particular, el caso $n = 3$ sería $u(x, y, z + \delta, t) = u_{z+\delta}$ o $u(x + \delta, y, z, t + \delta) = u_{x+\delta,t+\delta}$.

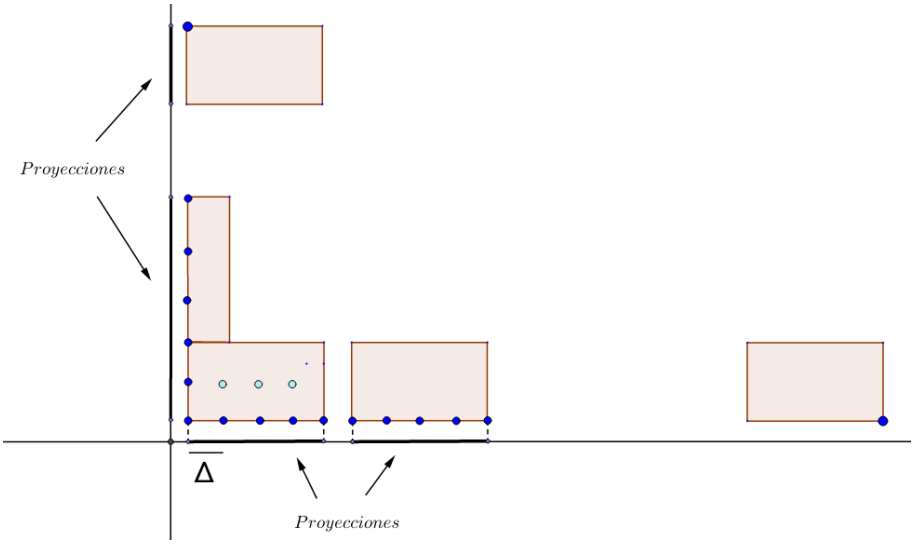



Figura 1: Ejemplo de descretización de un dominio en 2D. Los puntos más oscuros representan los puntos los puntos de la frontera y los puntos más claros son los puntos del interior del dominio finito.

Siendo $u(x_1, ..., x_n, t)$ la temperatura en un punto cualquiera del dominio $D_t$, se puede calcular, utilizando la expansión de Taylor en el punto $(x_1, ..., x_n, t) \in \Omega \times [0, 1]$, una aproximación de $u_{x_i+\delta} = u(x_i + \delta x_i, t)$.

$$u_{x_i+\delta} = u + \frac{\partial u}{\partial x_i}\delta + \frac{1}{2}\frac{\partial^2 u}{\partial^2 x_i}\delta^2 + \frac{1}{3!}\frac{\partial^3 u}{\partial^3 x_i}\delta^3 + O(\delta^4)$$

$$u_{x_i-\delta} = u - \frac{\partial u}{\partial x_i}\delta + \frac{1}{2}\frac{\partial^2 u}{\partial^2 x_i}\delta^2 - \frac{1}{3!}\frac{\partial^3 u}{\partial^3 x_i}\delta^3 + O(\delta^4)$$

$$\Rightarrow u_{x_i+\delta} + u_{x_i-\delta} = 2u + \frac{\partial^2 u}{\partial^2 x_i}\delta^2 + O(\delta^4) \Rightarrow$$

$$\Rightarrow \frac{\partial^2 u}{\partial^2 x_i} = \frac{u(x_i + \delta x_i, t) + u(x_i - \delta x_i, t) - 2u}{\delta^2} + O(\delta^2) \Rightarrow$$

$$\Rightarrow \sum_i^n \frac{\partial^2 u}{\partial^2 x_i} \approx \frac{1}{\delta^2}\sum_i^n [u(x_i + \delta, t) + u(x_i - \delta, t) - 2u(x, t)] + O(\delta^2)$$

Como se puede observar la aproximación de $\Delta u$ que se obtiene, arrastra un error de orden $O(\delta^2)$. Por lo tanto ahora se conocen todos los términos de la derecha de la ecuación diferencial ordinaria 3.1.

El siguiente paso del método es aproximar la última incógnita de la relación diferencial que queda por conocer, $u_t$. Por esto se plantea introducir métodos de resolución de ecuación diferenciales ordinarias, ya que 3.2 puede re-escribirse la ecuación diferencial matricial ordinaria como:

$$u_t = f(u, t)$$

En este punto dependiendo del método que se utilice, se puede encontrar la solución explícitamente, aunque no será estable dadas su propiedades. En cambio, si el método empleado es implícito obtendremos un sistema de ecuaciones lineales, ya que nos encontraremos con un número de incógnitas finito que queremos aproximar en tiempo $t$, que dependen unas de las otras por el hecho de que la temperatura de un punto dependa de los más cercanos. Todo esto acaba resultando en que $f(u, t)$ puede representarse con la forma matricial $Bu + c$, y así, para encontrar la solución para todo tiempo $t$, se necesita resolver un sistema de ecuaciones lineales de la forma:

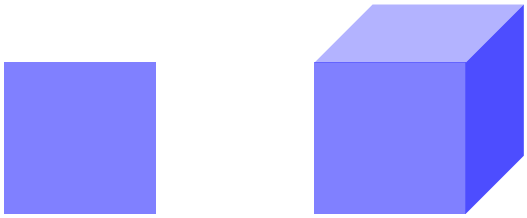
$$Au = b.$$

Habiendo solucionado el sistema lienal hayando la solución $u^*$, se tendrá una aproximación de la temperatura para cada punto de nuestro dominio y para todo tiempo $t \in [0, 1]$. De esta manera se podrá analizar los resultados para poder aplicarlo a otras condiciones iniciales y de frontera.

## 3.1. Aplicación a nuestros ejemplos

Para hacerse una mejor idea del método, cada paso se irá plasmando con un ejemplo de un dominio simple, descrito como producto cartesiano del intervalo $[0, 1]$ en las diferentes coordenadas del espacio, en dimensiones $n = 2, 3$.

Para la descretizacióne se segue la lógica planteada anteriormente donde se convenientemente $\delta = 10^{-3}$, por ejemplo, dividiendo el intervalo $[0, 1]$ en $N_i = 10^3 \;\; i \in \{x, y, z\}$ sub-intervalos, de esta medida. Como puntos de nuestro dominio, tomaremos los extremos de estos intervalos. Obtendremos para cada eje $i \in \{x, y, z\}$ $N_i + 1$ puntos que representen el dominio, que se denotará como $\Omega_F$, con cardinal $\#\Omega_F = \prod_{i\in\{1,\dots,n\}}(N_i + 1)$.



$[0,1] \times [0,1] \subset \mathbb{R}^2$. $[0,1] \times [0,1] \times [0,1] \subset \mathbb{R}^3$.

Figura 2: Representación de los doniminios utilizados en 2 y 3 dimensiones

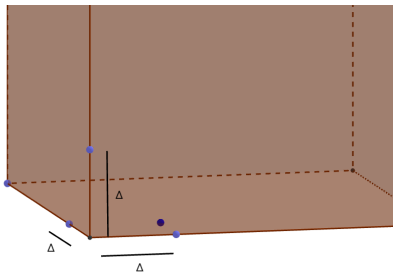


Figura 3: Descretización de parte del dominio espacial

Para entender mejor el proceso y para enfocar desde el principio el problema como una relación lineal, se reescribirá el problema como tal. Para ello hay que tener en cuenta que, para cada tiempo $t$, de los $\prod(N_i + 1)$ puntos de nuestro dominio $\Omega_F$, sólo $\prod(N_i - 1)$ puntos son incógnitas del sistema, mientras que los demás puntos pertenecerán a la frontera (donde la condición de Drichlet nos determina la temperatura para todo $t$). Se pondrán las incógnitas en forma de vector teniendo en cuenta el orden, como por ejemplo:

$$U(t) = \begin{pmatrix} u_{1,1,1} \\ \vdots \\ u_{1,1,N_x-1} \\ u_{1,2,1} \\ \vdots \\ u_{N_z-1N_z-1,N_z-1} \end{pmatrix} = \begin{pmatrix} U_0 \\ \vdots \\ U_{N_x-1} \\ U_{N_x} \\ \vdots \\ U_{N_xN_yN_z} \end{pmatrix}$$

Los puntos de la frontera, serán de la forma $u_{i,j,k}$ teniendo alguno de sus índices $0, N_x, N_y, N_z$. Dado un punto $(i\delta, j\delta, k\delta) \in D_t$ del dominio, se definen la temperaturas de los puntos que me referiré como vecinos más cercanos a un punto:

$$vecinos(i,j,k) := \{u_{i\pm1,j,k}, u_{i,j\pm1,k}, u_{i,j,k\pm1}\}$$

Se puede ver por la ordenación del vector que cada $u_{ijk}$ tiene asociado un $U_l$, por lo tanto a cada $(i,j,k)$ le corresponde un $l$. Teniendo esto en cuenta se definirá un $b_{Front}$ que acumulará, en la fila $l$ la temperatura de los puntos más cercanos que

también sean de la frontera:

$$b_{Front,l}(u_{ijk}) = \sum_{v \in vecinos(i,j,k) \cap \partial D_t} v$$

Hasta ahora, con lo que se ha visto, se puede aproximar el laplaciano de la siguiente forma:

$$\tilde{\Delta u} = \frac{-1}{\delta^2} \left( \sum_i^n -u(x_i + \delta, t) + 2u(x,t) - u(x_i - \delta, t) \right)$$

Para expresarlo de forma matricial se tiene que tener en cuenta que $u_{ijk}$ son incógnitas para añadirlo a la matriz del sistema, y cuales no lo son para ponerlo en forma de vector. A la hora de tratar el producto de matrices en el entorno de programación no será necesario guardar los coeficientes de la matriz $A$ en ninguna variable $n$-dimensional. En su lugar se tendrán escribirán explícitamente con una lógica de *if*, ya que como mucho se tendrán $2n$ coeficientes y todos serán iguales. Se puede resumir en que se sumarán los índices correspondientes a los $2n$ vecinos, excepto en caso de que sean frontera. El resultado de está ordenación de índices y en consecuencia de la elección de incógnitas no lleva a considerar a la familia de matrices tridiagonales a bloques de la siguiente forma:

$$T_1(a) = \begin{pmatrix} a & -1 & 0 & 0 & ... & 0 \\ -1 & a & -1 & 0 & ... & 0 \\ 0 & -1 & a & -1 & \ddots & \vdots \\ 0 & 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & ... & & \ddots & \ddots & -1 \\ 0 & ... & 0 & 0 & -1 & a \end{pmatrix} \in \mathbb{R}^{N_x - 1} \times \mathbb{R}^{N_x - 1}$$

$$T_2(a) = \begin{pmatrix} T_1 & I_{N_x-1} & 0 & 0 & ... & 0 \\ I_{N_x-1} & T_1 & I_{N_x-1} & 0 & ... & 0 \\ 0 & I_{N_x-1} & T_1 & I_{N_x-1} & \ddots & \vdots \\ 0 & 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & ... & & \ddots & \ddots & I_{N_x-1} \\ 0 & ... & 0 & 0 & I_{N_x-1} & T_1 \end{pmatrix} \in \mathbb{R}^{(N_y-1)(N_x-1)} \times \mathbb{R}^{(N_y-1)(N_x-1)}$$

$$T_3(a) = \begin{pmatrix} T_2 & I_{(N_y-1)(N_x-1)} & 0 & ... & 0 \\ I_{(N_y-1)(N_x-1)} & T_2 & I_{(N_y-1)(N_x-1)} & ... & 0 \\ 0 & I_{(N_y-1)(N_x-1)} & T_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & I_{(N_y-1)(N_x-1)} \\ 0 & ... & 0 & I_{(N_y-1)(N_x-1)} & T_2 \end{pmatrix}$$

$$T_3(a) \in \mathbb{R}^{(N_x-1)(N_y-1)(N_x-1)} \times \mathbb{R}^{(N_x-1)(N_y-1)(N_x-1)}$$

Ahora si se puede expresar la aproximación de $\Delta u$ en forma matricial, para obtener la ecuación diferencial ordinaria:

$$\tilde{\Delta u} = -\frac{1}{\Delta^2}(T_{dim}(2n)U_t + b_{Front})$$

que se denominará la matriz laplaciana de la función $u$, por lo tanto se tiene la siguiente representación de la relación diferencial:

$$\Rightarrow u_t \approx g(u,t) - \frac{1}{\delta^2}(T_{dim}(2n)U_t + b_{Front})$$

Para resolver la ecuación diferencial ordinaria se utilizarán los métodos de Euler implícito, Euler explícito y Crank-Nicolson. Los diferentes métodos, al aproximar la derivada de la temperatura respecto al tiempo, tendrá un efecto en los coeficientes de la aproximación (expresado de forma matricial). Los métodos explícitos darán, en general, una peor aproximación de la solución pero se tendrá una fórmula explícita de la solución, luego no será un método iterativo y se resolverá con el número de asignaciones igual a la dimensión de nuestras $(N_x - 1)(N_y - 1)(N_z - 1)^3 = (10^3 - 1)^3$ incógnitas. Para los métodos implícitos obtendremos para cada tiempo $t \in [0, 1]$ un sistema lineal para solucionar. Con esto discretizaremos el tiempo y encontraremos un sistema a tiempo $t$ que podremos solucionar en el siguiente paso si no lo solucionamos en este.

Los métodos que se utilizarán para resolver estos sistemas de ecuaciones lineales serán Jacobi, Gauss-Seidel y SOR y por último se mejorará la velocidad con el método del Gradiente conjugado. Se estudiará la convergencia, las iteraciones y el error de cada método para encontrar el óptimo para esta situación para verificar la validez de la solución y poder utilizarlo a otros casos.

# 4. Resolución numérica de ecuaciones diferenciales ordinarias

En este apartado se estudiará métodos explícitos e implícitos de sistema de ecuaciones diferenciales ordinadrias, partiendo de la relación $\dot{u} = f(t, x)$, definda en el dominio presentado en la sección anterior y suponiendo que conocemos las hipótesis de Dirichlet. Se empezará a presentar una serie de métodos que se estudian tipicamente en las universidades (en particular en la UB) como puede ser Euler Explícito y Euler Implícito. En la sección que le precede se conocerá el método de Crank-Nicolson motivado por querer encontrar un método que minimice el error hasta el mismo orden que la aproximación de $\Delta u$ y siguiendo la misma lógica que en los métodos anteriores, se verá como este método finalmente aproximará mejor la solución. Para todos ellos se estudiará la estabailidad y convergencia de estos métodos igual que la fórmula de error para poder compararlos y se verá la aplicación de estos para la ecuación diferencial que se presenta en este trabajo y se comprobará con práctica qué método es mejor.

## 4.1. Métodos básicos: Euler Explícito e Implícito

El método de Euler explícito encuentra una aproximación para cada punto cogiendo la definición de derivada por la derecha de la función $u$ respecto a la variable $t$, esto es:

$$\forall U \in \mathbb{R}^n \quad U_t(t) = \lim_{\delta_t \to 0} \frac{U(t+\delta_t) - U(t)}{\delta_t}$$

Asumimos que es una aproximación para obterner:

$$\forall U \in \mathbb{R}^n \quad f(u,t) = U_t(t) \approx \frac{U(t+\delta t) - U(t)}{\delta t}$$

A partir de aquí, se obtiene una expresión de $U(t+\delta_t)$, es decir la temperatura para el siguiente paso, aislándolo:

$$U(t+\delta_t) \approx U(t) + \delta_t \cdot f(u,t)$$

La estabilidad de este método se estudiará utilizando la ecuación test $u_t = \lambda u$, con $\lambda < 0$ y solución $C \exp^{\lambda t}$ . Se encontrará una expresión para el iterado $k$-ésimo $u^{(k)}$ con $k \to \infty$

$$u^{(k)} = u^{(k-1)} + \delta_t f(u^{(k-1)}, t^{(k-1)}) = u^{(k-1)}(1+\delta_t\lambda) = u^{(0)}(1+\lambda\delta_t)^k$$

$$\lim_{k\to\infty} u^{(0)}(1+\lambda\delta_t)^k = 0 \iff |1+\lambda\delta_t| < 1$$

$$|1+\lambda\delta_t| < 1 \Leftrightarrow -2 < \lambda\delta_t < 0, \quad \delta_t > 0 \Leftrightarrow \delta_t < \frac{2}{|\lambda|}$$

Por lo tanto el método no es estable y en caso particular en el que nos encontraremos que el error tiende a infinito a medida que $t$ tiende a infinito.

El método de Euler implícito empieza por coger la definición de derivada por la izquierda de la función $u$ respecto a la variable $t$ en el punto $t + \delta t$, esto es:

$$\forall t \in \mathbb{R} \quad \lim_{\delta_t \to 0} U_t(t + \delta t) = \lim_{\delta_t \to 0} \frac{U(t + \delta_t) - U(t)}{\delta_t}$$

Asumimos que es una aproximación para obtener:

$$\forall U \in \mathbb{R}^n \quad f(U, t + \delta_t) = U_t(t + \delta_t) \approx \frac{U(t) - U(t + \delta_t)}{\delta_t}$$

A partir de aquí, se obtiene una expresión implícita de $U(t + \delta_t)$ aislándolo, para encontrarse con una expresión implícita de la solución:

$$U(t + \delta_t) = U(t) + \delta_t \cdot f(U, t + \delta_t)$$

De igual manera que el caso anterior se utilizará la función test para comprobar su estabilidad:

$$u^{(k)} = u^{(k-1)} + \delta_t f(u^{(k)}, t^{(k)}) = u^{(k-1)} + \delta_t \lambda u^{(k)} \Rightarrow u^{(k)} = \frac{u^{(k-1)}}{1 - \lambda\delta_t} = \frac{u^{(0)}}{(1 - \lambda\delta_t)^k}$$

$$\lim_{k \to \infty} \frac{u^{(0)}}{(1 - \lambda\delta_t)^k} = 0 \iff \frac{1}{1 - \lambda\delta_t} < 1$$

Tenemos $\lambda < 0, \delta_t > 0$:

$$\frac{1}{1 - \lambda h_t} < 1 \Longrightarrow 1 < 1 - \lambda h_t \Longrightarrow 0 < -\lambda\delta_t = |\lambda|\delta_t$$

Observamos que no depende del valor de $\lambda$, por lo tanto podemos concluir que el método es estable.

Ahora calcularemos el error restando la solución exacta de la aproximada, utilizando la expansión de Taylor:

$$\frac{u(t + \delta_t) - u(t)}{\delta_t} = u'(t_n) + u''(t_n)\frac{\delta_t}{2} + O(\delta_t^2)$$

y teniendo en cuenta

$$u'(t + \delta_t) - u'(t) = u''(t)\delta_t + \frac{u'''(t)}{2}\delta_t^2 + O(\delta_t^3)$$

se obtiene que

$$f(u, t + \delta_t) - \frac{u_{n+1} - u_n}{\delta_t} = u'(t + \delta_t) - u'(t_n) - u''(t_n)\frac{\delta_t}{2} + O(\delta_t^2) =$$

$$= \frac{u''(t)}{2}\delta_t + O(\delta_t^2) := \epsilon$$

El orden de la función error es del orden de $\delta_t$, el mismo orden que el método de Euler explícito.

## 4.2. Crank-Nicolson

Hasta ahora la aproximación del laplaciano que se ha planteado es de orden $\delta^2$ mientras que los métodos de aproximación de la ecuación diferencial ordinaria que se han visto hasta ahora son de orden $\delta_t$. Tomando desde ahora $\delta_t = \delta$, se tendrá el problema de que las dos aproximaciones son de ordenes diferentes. Por lo tanto hay que encontrar un método que tenga un error de orden $\delta^2$, para que tenga sentido aproximar a orden $\delta^2$ el laplaciano $\delta u$. Para ello aproximaremos, con un error $\epsilon$ la derivada respecto al tiempo en $u$ con las funciones $f(t^{(k)}, u^{(k)})$ y $f(t^{(k+1)}, u^{(k+1)})$ y pesos $\theta, 1-\theta$ de la siguiente manera:

$$\frac{u^{(k+1)} - u^{(k)}}{\delta} \approx \theta f(t^{(k+1)}, u^{(k+1)}) + (1-\theta) f(t^{(k)}, u^{(k)})$$

Minimizaremos el error intentando eliminar los términos de orden $\delta$. Primero utilizando Taylor sabemos que:

$$u(t^{(k+1)}) - u(t^{(k)}) = u(t^{(k+1)} + \delta) - u(t^{(k)}) = u'(t^{(k)})\delta + u''(t^{(k)})\frac{\delta^2}{2} + o(\delta^3)$$

$$\Longrightarrow \frac{u^{(k+1)} - u^{(k)}}{\delta} = u'(t^{(k)}) + u''(t^{(k)})\frac{\delta}{2} + o(\delta^2)$$

Por otro lado tenemos que:

$$\theta f(t^{(k+1)}, u(t^{(k+1)})) + (1-\theta) f(t^{(k)}, u(t^{(k)})) = \theta u'(t^{(k+1)}) + (1-\theta) u'(t^{(k)}) =$$

$$= u'(t^{(k)}) + \theta(u'(t^{(k+1)}) - u'(t^{(k)})) = u'(t^{(k)}) + \theta\delta \cdot u''(t^{(k)}) + o(\delta^2)$$

Si se restan los términos anteriores, se tiene que el error es el siguiente:

$$\theta f(t^{(k+1)}, u^{(k+1)}) + (1-\theta) f(t^{(k)}, u^{(k)}) - \frac{u(t^{(k)}) - u(t^{(k)})}{\delta} =$$

$$= \delta \cdot u''(t^{(k)})(\theta - \frac{1}{2}) + o(\delta^2) := \epsilon_k$$

Observamos que el orden del método es $o(\Delta^2)$ para $\theta = \frac{1}{2}$, mientras que para $\theta \neq \frac{1}{2}$ el orden del método es $o(\delta)$. Si $\theta = \frac{1}{2}$, llamaremos al método resultante el método de Crank-Nicolson obteniendo la siguiente expresión:

$$\frac{U_{t+\delta} - U_t}{\delta} = \frac{f(u,t) + f(u, t+\delta)}{2} + \epsilon$$

De igual manera veremos la estabilidad del método utlizando una vez más la función test para un $\theta$ cualquiera, para el sistema:

$$\begin{cases} u' = -\lambda y, & t \geq 0, \\ u(0) = u_0 \end{cases}$$

Los cáculos anteriores nos dicen que:

$$u_{n+1} = \frac{1 - (1-\theta)\lambda\delta}{1 + \theta\lambda\delta} u^{(k)}$$

definiendo

$$r(x) := \frac{1-(1-\theta)x}{1+\theta x}$$

tenemos que:

$$u^{(k)} = r(\lambda\delta)^k u^{(0)}$$

hay que estudiar $|r(\lambda\delta)| \leq 1$. Observamos que esto ocurre para todo $\delta$ si $\theta \geq \frac{1}{2}$, en cambio si $\theta < \frac{1}{2}$ solo se verifica si:

$$\lambda\delta \leq \frac{2}{1-2\theta}$$

Por lo que podemos concluir que el método será incondicionalmente estable si $\theta \geq \frac{1}{2}$. Será condicionalmente estable si $0 \leq \theta < \frac{1}{2}$

Notamos también que si $\theta = 0$ entonces responde al método de Euler Explícito, si tenemos $\theta = 1$ obtendremos el método de Euler Implícito. Por último, si $\theta = \frac{1}{2}$ tendremos el método de Crank-Nicolson (el uńico de orden 2).

## 4.3. Aplicación al ejemplo

Se ha visto como teniendo una aproximación $\tilde{\Delta u}$, la relación diferencial inicial se resume en la siguiente ecuación diferencial ordinaria:

$$u_t = f(u,t) := g(u,t) - \frac{1}{\delta^2}(T_{dim}(2n)U(t) + b_{Front})$$

Nos planteamos utilizar los métodos explicados a los dominios $\Omega_F$ de dos y tres dimensiones que se han descrito en capítulo 3, y se tomará $\delta_t = \delta$

- Explícto :

$$\forall x \in \mathbb{R}^n \quad u_t(x,t) = \lim_{\delta \to 0} \frac{u(x,t+\delta) - u(x,t)}{\delta} \Rightarrow$$

$$\Rightarrow \tilde{U_t} = \frac{U_{t+\delta} - U_t}{\delta} = f(u,t) \Rightarrow$$

$$\Rightarrow U_{t+\delta_t} = U_t + \delta f(u,t) =$$

$$= U_t + \delta[g(u,t) - \frac{1}{\delta^2}(T_n(2n)U_t + b_{Front})] \Rightarrow$$

$$\Rightarrow U_{t+\delta} = U_t + \delta g(x,t) - \frac{1}{\delta}(T_n(2n)U_t + b_{Front})$$

$$U_{t+\delta} = b$$

- Implícito :

$$\tilde{U_{t+\delta}} = \frac{U_{t+\delta} - U_t}{\delta} = f(u,t+\delta_t) \Rightarrow$$

$$\Rightarrow U_{t+\delta} = U_t + \delta f(x,t+\delta) =$$

$$= U_t + \delta g(u,t+\delta) - \frac{1}{h}(T_n(2n)U_{t+\delta} + b_{Front})$$

$$\Rightarrow (I + \frac{1}{\delta}T_n(2n))U_{t+\delta} = U_t + \delta g(u,t+\delta) - \frac{1}{\delta}b_{Front}$$

$$\Rightarrow AU_{t+\delta} = b$$

- Crank-Nicolson :

$$\frac{f(u,t) + f(u,t+\delta)}{2} = \frac{U_{t+\delta}}{\delta} \Rightarrow$$

$$\frac{\delta}{2}[g(u,t) - \frac{1}{\delta^2}(T_n(2n)U_t + b_{tFront}) + g(u,t+\delta) - \frac{1}{\delta^2}(T_n(2n)U_{t+\delta} + b_{(t+\delta)Front})] = U_{t+\delta} - U_t$$

$$U_{t+h} + \frac{1}{2h}T_{dim}(2n)U_{t+h} = U_t - \frac{1}{2\delta}(T_n(2n)U_t + b_{tFront} + b_{(t+\delta)Front}) + \frac{\delta}{2}(g(u,t) + g(u,t+\delta))$$

$$(I + \frac{1}{2\delta}T_{dim}(2n))U_{t+\delta} = U_t - \frac{1}{2\delta}(T_{dim}(2n)U_t + b_{tFront} + b_{(t+\delta)Front}) + \frac{\delta}{2}(g(u,t) + g(u,t+\delta))$$

$$\Rightarrow AU_{t+\delta} = b$$

Se puede ver como aplicando Euler explícito ya tenemos la aproximación para la temperatura en todo el dominio espacio-temporal. Esto significa que podemos ver el error que tiene el método numéricamente. A continuación una tabla con la diferencia entre la solución real,fijada a priori, $u^*(x,y,z,t) := \cos(x+y+z+t)$ y la aproximada por el Euler explícito $\bar{u}(x,y,z,t)$. Para ello tambien se tendrá en cuenta la función $g(u(x,y,z,t),t) = u_t((x,y,z,t)) - \Delta u(x,y,z,t) = 3\cos(x+y+z+t) - \sin(x+y+z+t)$ como una de las hipótesis que deberíamos conocer.

| tiempo | **Error máximo en Euler Explícito** |
|---|---|
| $t = 0{,}01$ | $5{,}022 \times 10^{-5}$ |
| $t = 0{,}02$ | $1{,}497 \times 10^{-2}$ |
| $t = 0{,}03$ | $5{,}985$ |
| $t = 0{,}04$ | $2{,}840 \times 10^{3}$ |
| $t = 0{,}05$ | $1{,}523 \times 10^{6}$ |
| $t = 0{,}06$ | $8{,}984 \times 10^{8}$ |
| $t = 0{,}07$ | $5{,}720 \times 10^{11}$ |
| $t = 0{,}08$ | $3{,}881 \times 10^{14}$ |
| $t = 0{,}09$ | $2{,}779 \times 10^{17}$ |
| $t = 0{,}1$ | $2{,}083 \times 10^{20}$ |

Tabla 1: Tabla de Error de Euler explícito en tres dimensiones

Con este método es una asignación directa de la temperatura para cada punto del dominio espacio-temporal finito. En cambio los métodos implícitos que se obtienen resultan en un sistema lineal de dimensión $N_x \times N_y = 10^6$ o $N_x \times N_y \times N_z = 10^9$ por cada punto del dominio,es decir, el número de asignaciones es mucho mayor. Como es un método inestable los resultados que obtenemos en la primera paso de tiempo es bastante buena, pero a medida que pasa el tiempo el error se va haciendo grande en pocos iterados, tendiendo al infinito cómo se esperaba. Se puede observar como tomando un paso de $\delta_t = 0{,}01$, en el octavo $t = 0{,}1$ ya tiene un error de $2{,}083 \times 10^{20}$ por lo que no tiene sentido utilizar este método. Este tipo de comportamiento viene por que el método es inestable y tiene que cumplir una seres de condiciones que a medida que aumenta la dimensión aumenta el número de condiciones. Por ello en la resolución de EDPs se pasa de un problema de dimensión infinita a uno finito, esto produce que las dimensiones suelan ser cuanto más grandes mejor y hace que pierda el sentido utilizar Euler. Por ello no se utilizará en ninguno de los casos.

# 5. Resolución numérica de sistemas lineales

Como hemos visto, utilizar un método implícito para solucionar la EDO conlleva resolver un sistema lineal $Ax = b$ . Hay varios grupos de métodos de resolución lineal, si los métodos no iterativos, requerirá una manipulación de la matriz y necesitará ser guardada en su completitud, aunque siempre son convergentes mientras que los iterativos tienen que cumplir condiciones asociadas a cada método. Los métodos iterativos no requieren de ese espacio de memoria en la máquina si se pueden capturar los coeficientes con cierto patrón. Esto se podría complicar si la matriz tiene muchos coeficientes diferentes a cero, pero en el caso que nos planteamos será una matriz sparse donde cada fila tendrá lo sumo $2n$ y por lo tanto no le daremos importancia . Por ello, en este capítulo se resumen los métodos iterativos de resolución de sistemas lineales de Jacobi, Gauss-Seidel y SOR, para compararlos con el método del Gradiente Conjugado, un método que plantea el problema como uno de minimización. De esta manera encontraremos la aproximación de la solución que se busca, para finalmente, con un ejemplo, se ilustrarán las diferencias de los diversos métodos desde el punto de vista de la aproximación a la solución real y el número de iteraciones para llegar a converger a ella.

## 5.1. Métodos iterativos básicos

A continuación repasaremos los métodos más sencillos que resuelven un sistema lineal $Ax = b$, con $A \in \mathbb{R}^n \times \mathbb{R}^n$ y $x, b \in \mathbb{R}^n$ descomponiendo la matriz como $A = L + D + U$, donde $L$ es la triangular inferior, $D$ es la diagonal, con todo elemento diferente de cero, y $U$ es la triangular superior. Hay que tener en cuenta que $n$ representa la dimensión del sistema lineal, y no la dimensión del dominio como en troas secciones. Los métodos de Jacobi y Gauss-Seidel basan su razonamiento 'aislar' cada componente de $x$ para así, de una manera recursiva, encontrar una aproximación que tiende a la solución real. El método de SOR, parte del de Gauss-Seidel y aplica un termino de sobrerelajación, que permite mejorar los métodos anteriores. Se estudiará el error de cada método expresando el sistema de la siguiente forma:

$$x^{(k+1)} = Bx^k + c \tag{5.1}$$

con $B \in \mathbf{R}^n \times \mathbf{R}^n$ y $c \in \mathbf{R}^n$. Si denotamos $x_*$ como la solución del sistema, esta cumple $x_* = Bx_* + c$. Si restamos estas dos igualdades y definimos:

$$e^{(k+1)} := x^{(k+1)} - x_*$$

se puede tomar una norma matricial consistente para obterner:

$$(x^{(k+1)} - x_*) = B(x^{(k)} - x_*) \Rightarrow$$

$$||e^{(k+1)}|| = ||Be^{(k)}|| < ||B^{k+1}||||e^{(0)}|| < ||B||^{k+1}||e^{(0)}||$$

Se observa que el sistema convergera si

$$\lim_{k\to\infty} \|B\|^k = 0 \iff ||B|| < 1$$

A continuación un breve recordatorio de los métodos numéricos de resolución de sistemas lineales que se han estudiado a lo largo de la carrera, expresados con la forma (5.1).

**Jacobi:** Suponemos que la diagonal $D$ del sistema no tiene elementos nulos.

$$Ax = (L + D + U)x = b$$

$$x^{(k+1)} = D^{-1}((U + L)x^{(k)} - b)$$

Luego, con la notación anterior se tien que $B = D^{-1}(U + L)$ y $c = D^{-1}b$

**Gauss-Seidel:** Suponiendo que aproximamos $x_i$ con el orden $x_1, x_2, ..., x_n$ Para aproximar las componentes $x_i^{(k+1)}$ del vector $x$, siendo $k$ el paso, se utilizan los elementos $x_j^{(k+1)}$, $j < i$, ya calculados del paso actual. Esto es:

$$Ax = (L + D + U)x = b$$

$$x^{(k+1)} = -L(D + U)^{-1}x^{(k)} + (D + U)^{-1}b$$

luego $B = -L(D + U)^{-1}$, y $c = (D + U)^{-1}b$

**SOR:** Se trata de utilizar el método anterior y expresarlo de la forma $x^{(k+1)} = x^k + \omega F(A, x, b)$ e influir con $w \in (0, 2) \subset \mathbb{R}$ en la convergencia del método de Gauss-Seidel. Primero re-escribamos el método anterior sumando y restando $x_i^{(k)}$, de la siguiente forma:

$$x_i^{(k+1)} = x_i^{(k)} - \frac{1}{a_{i,i}}(\sum_{j<i} a_{i,j}x_j^{(k+1)} + \sum_{j\geq i} a_{i,j}x_j^{(k)} - b)$$

Ahora se añade el término $w \neq 0$ como término de sobre-relajación del método anterior:

$$x_i^{(k+1)} = x_i^{(k)} - \frac{w}{a_{i,i}}(\sum_{j<i} a_{i,j}x_j^{(k+1)} + \sum_{j\geq i} a_{i,j}x_j^{(k)} - b)$$

Aunque a la hora de programar el método puede ser más conveniente ver el sistema de esta forma, expresado como en 5.1 sería:

$$(D + wL)x_i^{(k+1)} = wb - [wU + (w - 1)D]x_i^{(k)}$$

Luego $B = -(D + wL)^{-1}(wU + (w - I)D)$, y $c = (D + wL)^{-1}wb$

Se ha visto como cuando $w = 1$ el sistema es el de Gauss-Seidel y se intenta encontrar un valor óptimo en el intervalo abierto $(0, 2)$, luego podría ser 1. Se puede calcular una aproximación $w^* \in (0, 2)$ del $\bar{w}$ óptimo de varías maneras en el caso que estamos. Podríamos calcular el radio espectral de la matriz $B$ descrita en el método de Gauss-Seidel o S.O.R.. Con este valor se obtiene $w$ con la siguiente igualdad, ya que nuestra matriz es simétrica, definida positiva, con coeficientes reales y tridiagonal a bloques :

$$\bar{w} = \frac{2}{1 + \sqrt{(1 - \rho(B)}}$$

En los ejemplos se utiliza otro método ya que no tendremos siempre matrices tridiagonales a bloques (aunque cumplirán las otras hipótesis), consiste en comparar las soluciones aproximadas con las reales, fijada la función $u$ y calculando los valores de $u_t$, $\Delta u$, y $g(X,t)$. Con un paso de $h = 0{,}05$, obtenemos para cada $w \in \{h, 2h, ..., 20h\}$ la diferencia de las dos soluciones utilizando la norma $||\cdot||_\infty$. Por lo tanto se obtendrá una aproximación $w* = \bar{x} \pm h$.

## 5.2. Método del gradiente conjugado

En los métodos anteriores calcular los valores propios de la matriz $B$ asociada al sistema (para ver la convergencia, o calcular el $\omega$ óptimo) puede ser complicado. O quizás no son lo suficientemente rápido como queremos cuando las dimensiones empiezan a ser grande. Por ello se presenta un método para matrices reales definidas positivas y simétricas, que permite encontrar la aproximación de la solución más rápido. El método del Gradiente conjugado plantea el problema de encontrar la solución de $Ax - b = 0$ de forma diferente. Tomará una función $f(x) = \frac{1}{2}x^T Ax - x^T b,\ x \in \mathbb{R}^n$, cuyo mínimo será la solución del sistema, es decir lo transforma en un problema de minimización. La dirección a serguir se expresará en una base de vectores conjugados, a partir del gradiente en cada punto para así hayar la solución.

Supongamos que se tiene $\{p_i\}$ base ortogonal respecto al producto escalar $<,>$: $\mathbb{R}^n \times \mathbb{R}^n \longrightarrow \mathbb{R}$ definido como $< u^t, v >= u^t Av$ de $\mathbb{R}^n$ y $x_* = \sum_i^n \alpha_i p_i$ solución del sistema $Ax = b$ expresada en la base $\{p_i\}_i$. Entonces se pueden calcular los coeficientes $\alpha_i$ explicitamente de la siguiente manera:

$$b = Ax_* = \sum_i^n \alpha_i A p_i$$

$$p_k^T b = \sum_i^n \alpha_i p_k^T A p_i = \alpha_k p_k^T A p_k$$

$$\alpha_k = \frac{p_k^T b}{p_k^T A p_k} \tag{5.2}$$

Como a priori no se tienen los vectores $\{p_i\}$, se calculará la base ortogonal con el método de Grand-Smidth a partir del vector gradiente $r_k$ (también conocido como residuo), en cada punto $x_k$ y siendo $x_0$ el punto inicial, lo más aproximado a la solución posible. Luego el vector $p_1$ calculado no será ortogonal a $p_0 = r_0$, sino a un $\tilde{p_0} = r_1$, que no se necesitará ya que para calcular un $\alpha_k$ se requiere únicamente el vector $p_k$. Se deducirá cada vector $p_i$ de la base para hallar la solución $x_n$ de la siguiente forma:

$$r_k = b - Ax_k$$

$$p_k = r_k - \sum_{i<k} \frac{p_i^T A r_k}{p_i^T A p_i} p_i$$

$$x_{k+1} = x_k + \alpha_k p_k$$

Halladas todas las variable que intervienen en el planteamiento se ha conseguido una solución directa. A la práctica, en matrices de dimensiones grandes donde el número de iteraciones será tan grande que es suficiente acercarse a la solución con cierta tolerancia y por esta razón se plantea habitualmente como un método iterativo.

Con intención de hacer el mínimo de cálculos posibles se pone en función de $r_k, r_{k+1}$ , sabiendo que $r_k$ es ortogonal a $p_i$ con $i < k$, $r_k \cdot_i = 0$, y teniendo en cuenta el orden, se utilizan los términos de la siguiente forma:

$$p_k^T b = p_k^T r_{k-1} - p_k^T A x_{k-1} = p_k^T r_{k-1} =$$

$$= p_k^T (r_k + \alpha A p_{k-1}) = p_k^T r_k = (r_k - \sum_{i<k} \frac{p_i^T A r_k}{p_i^T A p_i} p_i)^T r_k = r_k^T r_k \iff$$

$$\iff \alpha_k = \frac{p_k^T b}{p_k^T A p_k} = \frac{r_k^T r_k}{p_k^T A p_k} \tag{5.3}$$

$$r_{k+1}^T A p_k = \frac{r_{k+1}^T (r_{k+1} - r_k)}{\alpha_k} = \frac{r_{k+1}^T r_{k+1}}{\alpha_k} \tag{5.4}$$

$$p_k^T A p_k = (r_k + \beta_k p_{k-1})^T A p_k = \frac{r_k^T (r_{k+1} - r_k)}{\alpha_k} = \frac{r_k^T r_k}{\alpha_k} \tag{5.5}$$

$$\beta_k = -\frac{r_{k+1}^T A p_k}{p_k^T A p_k} = \frac{-(\ 5.4)}{(\ 5.5)} = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}$$

Teniendo en cuenta la igualdad 5.2, la construcción de $\{p_i\}_i$ , y los cálculos descritos, se traduce al siguiente pseudo-código:

$$\begin{aligned}
&Inicializacion: \\
&\qquad r_0 = p_0 = b - A x_0; \\
&\qquad k = 0; \\
&while \qquad ||r_k|| < tolerancia : break \\
&\qquad \alpha_k = \frac{r_k^T r_k}{p_k^T A p_k} \\
&\qquad x_{k+1} = x_k + \alpha_k p_k \\
&\qquad r_{k+1} = b - A x_{k+1} \\
&if \quad ||r_k|| < tolerancia : \\
&\qquad \beta_k = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k} \\
&\qquad p_{k+1} = r_{k+1} + \beta_k p_k \\
&\qquad k = k + 1
\end{aligned}$$

Con esto se tiene una solución aproximada en el vector $x_N$ con $N \in \mathbf{N}$ . Sabemos que, como mucho, el método acabará en tantos pasos como la dimensión de $A$, ya que puede ser escrito como un método directo, por lo tanto convergirá aunque no sepas cuanto puede tardar. Como ya se ve en el código la tolerancia *tol* será el error que se asumirá.

## 5.3. Aplicación a un ejemplo

En este paso ya se tienen claro los coeficientes de la matriz $A$ del sistema dependiendo si se ha utilizado Euler implícito o Crank-Nicolson. Cuando se plantea resolver el sistema lineal, en el caso de los métodos más básicos (Jacobi, Gauss-Seidel y SOR) no tienen que cumplir ninguna hipótesis en especial y su convergencia se comprobará numéricamente (ya que solo nos importa este caso), por lo que no se calcularán valores propios de las matrices del método. En cambio para el método del Gradiente Conjugado se tiene que verificar que la propia matriz del sistema $A$ sea una matriz definida positiva para poder aplicarle el Gradiente Conjugado.

Estos métodos conllevan la multiplicación de matrices y vectores y eso puede resultar costoso para la máquina. Por suerte la manera en la que se ha planteado el problema se conocerá qué elementos corresponden para calcular el vector $i$-ésimo del vector solución (los $vecinos(i,j,k)$ que no sean frontera). Por este motivo los cálculos serán menores como y los métodos tardarán menos, tal y como se ha descrito antes. Teniendo en cuenta esto y que todos los coeficientes menos los de la diagonal son iguales, no hará falta multiplicarlas explícitamentey es aplicar los métodos anteriores con las matrices definidas en la últimas sección del capítulo anterior.

Ahora se comprobará si la matriz del sistema lineal presentado define un producto interior, es decir, es simétrica y definida positiva en el sentido estricto. Primero es fácil ver que l matriz tridiagonal a bloques $A$ es simétrica ya que todos elementos de la matriz, menos la diagonal, son iguales. Esto se traduce a que todos sus valores propios son reales, y hay que saber que estos valores son positivos en el sentido estricto.Utilizaremos el teorema de Gregorin para saber en qué unión de discos están contenidos los valores propios, este es:

**Teorema 5.1** (**Teorema Gerschgorin**). *Sea $A \in \mathbb{C}^n \times \mathbb{C}^n$ una matriz con coeficientes $A = \{a_{ij}\}_{i,j=1,...,n} \in \mathbb{C}$,y sea $\lambda_1, ..., \lambda_n$ sus valores propios asociados. Entonces $\lambda_1, ..., \lambda_n$ están contenidos en la unión de los discos $D_i$ del plano complejo, siendo:*

$$D_i = D(a_{ii}, r_i) = \{z \in \mathbb{C} : |z - a_{ii}| \leq r_i\}, \quad r_i = \sum_{j \neq i} a_{ij}$$

*Demostración.* Sean $v_1, v_2, ..., v_n$ los vectores asociados a los valores propios $\lambda_1, ..., \lambda_n$ respectivamente, entonces para cada $i = 1, ..., n$ se tiene:

$$\sum_{j=1}^{n} a_{ij} v_{ij} = \lambda_i v_i \iff \lambda_i v_i - a_{ii} v_i = \sum_{j=1, j \neq i}^{n} a_{ij} v_{ij} \iff$$

$$\iff |\lambda_i - a_{ii}| = |\frac{\sum_{j=1, j \neq i}^{n} a_{ij} v_{ij}}{v_i}| \leq \sum_{j=1, j \neq i}^{n} |\frac{a_{ij} v_{ij}}{v_i}| \leq \sum_{j \neq i} a_{ij} = r_i$$

Ahora si podemos asegurar que $\lambda_i \in D(a_{ii}, r_i) \forall i = 1, ..., n.$ #

Teniendo en cuenta que los elementos de la diagonal son iguales el centro de cada disco (en nuestro caso un intervalo) es el mismo, por lo tanto, si encontramos

el máximo de los radios nuestros valores propios estarán comprendidos en el ese intervalo. Se calcula el radio dada la matriz que tenemos en los casos de Euler implícito y Crank-Nicolson. En el caso de Euler la fila que más elementos contiene tiene el siguiente sumatorio en 2D y 3D respectivamente:

$$\max_{i\in\{1,..,n\}} r_i = \frac{4}{\delta}, \quad \max_{i\in\{1,..,n\}} r_i = \frac{6}{\delta}$$

$$D_{2D}(1+\frac{4}{\delta},\frac{4}{\delta}) \quad D_{3D}(1+\frac{6}{\delta},\frac{6}{\delta})$$

Siendo $i$ la dimensión del dominio y $D_{iD}(a,b)$ el disco con centro $a$ y radio $b$ donde están contenido los valores propios. Como se ha visto que los vaps son reales, los valores propios estarán contenido en intervalos cerrados. En estos casos viene dado por el intervalo $[1, 1+2r_{iDmax}]$

En el caso de Crank-Nicolson la fila que contiene todos los elementos y por lo tanto el radio máximo en 2D y 3D respectivamente da el sumatorio:

$$r_{2Dmax} = \frac{2}{\delta}, \quad r_{3Dmax} = \frac{3}{\delta}$$

$$D_{2D}(1+\frac{2}{\delta},\frac{2}{\delta}) \quad D_{3D}(1+\frac{3}{\delta},\frac{3}{\delta})$$

De forma parecida el intervalo cerrado para Crank-Nicolson es $[1, 1+2r_{iDmax}]$.

Por lo tanto todos los valores propios son reales y estrictamente positivos, luego nuestra $A$ es simétrica y definida positiva por lo que cumple todas las hipótesis planteadas en el método del gradiente conjugado y ya podemos aplicarlo.

Para aplicar el gradiente conjugado, convendrá tener una función que multiplique un vector por la matriz $A$ del sistema lineal, ya que esta no necesita ser guardada por su estructura y se necesita de esa multiplicación a diferencia que en los métodos más sencillos. En nuestro caso el método tendrá el mismo código que el descrito en 5.2, con la matriz $A$ que tengamos en cada caso (Euler Implícitoo Crank-Nicolson). Como ya sabemos las dimensiones de la matriz $A$ en este caso tendrá $(10^3-1)(10^3-1)$ y $(10^3-1)^2(10^3-1)$ columnas y filas en 2 y 3 dimensiones respectivamente.

## 5.4. Resultados y visualización

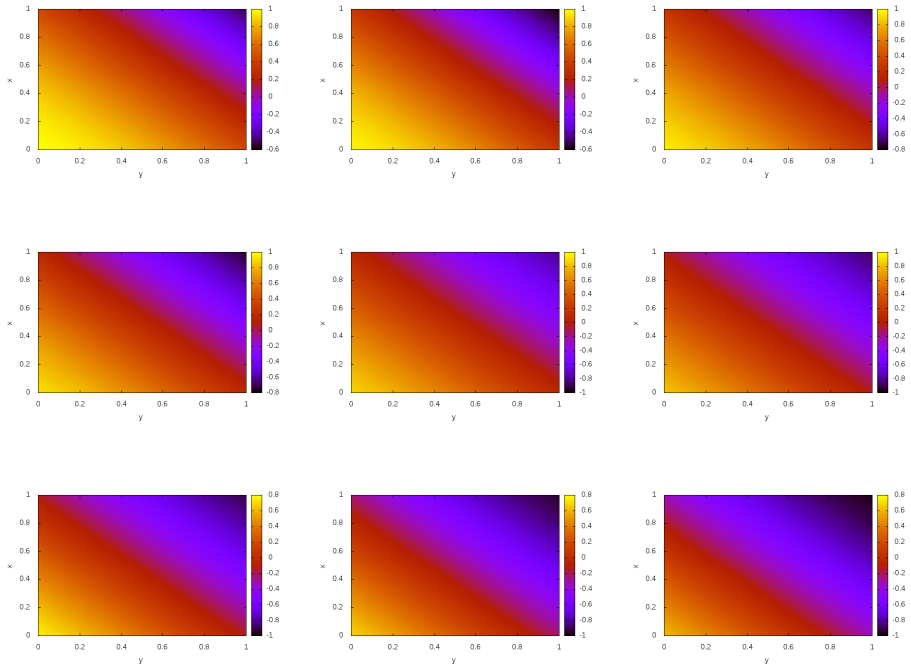
Ya se ha encontrado una aproximación y queremos validarla con una función fijada para poder extrapolar el programa a cualquier condición, La función que se ha fijado en este caso será $u(x,y,z,t) = \cos(x+y+z+t)$ y se ha deducido $\Delta u = -2n\cdot\cos(x+y+z+t)$, $u_t = -\sin(x+y+z+t)$ y $g(x,y,z,t) = 2n\cdot\cos(x+y+z+t) - \sin(x+y+z+t)$. Conociendo la solución real $u^*$ para cada punto, y la aproximación calculada $u$, se toma como error la máxima diferencia de los puntos del dominio para cada tiempo t (cada vez que se ejecuta el método).

| | | nº iteraciones | tiempo de ejecución | Error máximo |
|---|---|---|---|---|
| Jacobi | Implícito | 5903 | 451s | $1.676\times10^{-4}$ |
| | Crank-Nicolson | 3331 | 198s | $1.030\times10^{-6}$ |
| Gauss-Seidel | Implícito | 3070 | 171s | $1.676\times10^{-4}$ |
| | Crank-Nicolson | 1733 | 99s | $1.029\times10^{-6}$ |
| SOR | Implícito | 233 | 26s | $1.676\times10^{-4}$ |
| | Crank-Nicolson | 201 | 23s | $1.029\times10^{-6}$ |
| Gradiente Conjugado | Implícito | 183 | 6s | $1.676\times10^{-4}$ |
| | Crank-Nicolson | 139 | 4s | $1.028\ \times10^{-6}$ |

| | | nº iteraciones | tiempo de ejecución | Error máximo |
|---|---|---|---|---|
| Jacobi | Implícito | 8126 | 650m | $6.977\times10^{-5}$ |
| | Crank-Nicolson | 4740 | 400m | $6.317\times10^{-7}$ |
| Gauss-Seidel | Implícito | 4227 | 380m | $6.977\times10^{-5}$ |
| | Crank-Nicolson | 2465 | 200m | $6.392\times10^{-7}$ |
| SOR | Implícito | 312 | 93m | $6.977\times10^{-5}$ |
| | Crank-Nicolson | 292 | 66m | $6.311\times10^{-7}$ |
| Gradiente Conjugado | Implícito | 254 | 33m | $2.139\times10^{-4}$ |
| | Crank-Nicolson | 186 | 26m | $8.69\times10^{-7}$ |

Como es de esperar el caso de dos dimensiones es más rápido que el de tres, puesto que tiene menos incógnitas y a medida que el método es más sofisticado, menor es el número de iteraciones y el tiempo de ejecución. Si uno se fija en el error, se da cuenta de la importancia del orden en los métodos de ecuaciones deferenciales ordinarias, donde la diferencia entre los dos métodos es de $\times10^2$ aproximadamente. En particular, el método de S.O.R. requiere hacer la computación para cada $w \in (0, 2)$, en este caso se ha utilizado un paso de 0,05, como ya se ha comentado. Los datos de las tablas, en la fila "SOR", representan los resultados del caso en el que se ha considerado como $w$ óptimo los valores $1{,}75, 1{,}8, 1{,}85$ dependiendo del caso en del método y de la dimensión. Y se puede observar como realmente la sobre-relajación nos sirve para converger más rápido a la solución.

Para visualizar la respuesta ya conociendo el error y teniendo en cuenta como limitación, que es una aproximación de segundo orden. A continuación están representados el cambio de temperatura en dos dimensiones en tiempo $t = 0{,}1, 0{,}2, ..., 0{,}9 \in [0, 1]$. La visualización del dominio tridimensional representada en planos (pantalla) es más fácil de comprender en software dotado de visualización de capas dinámicas como se explicará en el apartado 7.2 para poder apreciar los cambios con el paso del tiempo de todas las proyecciones y el cambio de temperatura en el interior del dominio. Por lo tanto se escapa de lo planteado por este trabajo, no obstante a continuación se representa el cambio de temperatura del dominio dos dimensional en varias figuras para visualizar los resultados obtenidos utilizando como ejemplo la función mencionada $u(x, y, z, t) = \cos(x + y + z + t)$.



Utilizando el programa FFMPEG crearemos un vídeo juntando los *frames* de las soluciones aproximadas por el método y representadas gráficamente con el programa *gnuplot* . Siendo el *frame* $i$-ésimo la imagen con nombre salidai.png utilizamos el siguiente comando en la consola :

```
ffmpeg framerate 20 -i salida%d.png output.mp4
```

Estas herramientas pueden ir muy a la hora de utilizar otras condiciones y observar el resultado. Por ejemplo, se puede tener como condición inicial $u(x,0) = 0$ y como condición de frontera $u(x,t) = 100$. En este caso se podría observar con el paso del tiempo como tiende la temperatura a la condición de frontera 100.

# 6. Aplicación a otros dominios, Horno en 2D y 3D

A continuación se estudiará el mismo proceso, pero esta vez sobre otro dominio un poco más complejo que definiremos como un horno sellado, para plasmar su forma en pocas palabras. Estará formado por un cubo/cuadrado en 3D/2D, quitándole otro cubo/cuadrado más pequeño de su interior, dejando un anillo cúbico/rectangular (figura 4). Los dominios tendrán una cierta anchura que se denotará como $a$. En este caso las hipótesis de Dirichlet, que se conocerán a priori, cambiarán las dimensiones y los índices de las matrcices pero el proceso es el mismo. Se descretizarán utilizando la misma lógica que la explicada anteriormente y cuando llega la hora de solucionar la ecuación diferencial se utilizarán los métodos implícitos para entender las diferencias de orden entre solucionará y no el de Euler Explícito. El sistema lineal que se obtiene, se solucionará utilizando los métodos de SOR y Gradiente Conjugado para poder comprobar las posibles diferencias entre los métodos en dominios diferentes.

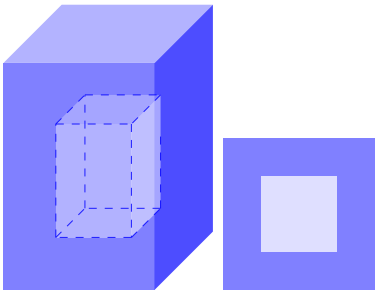


Figura 4: Representación del dominio referido como horno en $\mathbb{R}^3$ (izq.)y $\mathbb{R}^2$ (der.).

## 6.1. Discretización

El proceso para discretizar estos dominios será el mismo que en el apartado 1,2,1, por lo tanto se tomara:

$$\mathrm{X} \times Y = [0,1] \times [0,k]/[a,1-a] \times [a,k-a], \quad N_x = \frac{1}{\delta}, N_y = \frac{k}{\delta}$$

$$X \times Y \times Z = [0,1] \times [0,1] \times [0,k]/(a,1-a) \times (a,1-a) \times (a,k-a), \quad N_x, N_y = \frac{1}{\delta}, N_z = \frac{k}{\delta}$$

El dominio finito, que denotaremos como $\Omega_F$ resulta ser el siguiente:

$$\Omega_{2F} = \{(x,y) : (i \cdot \delta, j \cdot \delta), \quad (i,j) \in (X \times Y) \cap (\mathbb{N} \times \mathbb{N})\}$$

$$\Omega_{3F} = \{(x,y,z) : (i \cdot \delta, j \cdot \delta, h \cdot \delta), \quad (i,j,h) \in (X \times Y \times Z) \cap (\mathbb{N} \times \mathbb{N} \times \mathbb{N})\}$$

con cardinales respectivos:

$$\#\Omega_{2F} = (10^3-1)(k \cdot 10^3-1)-(10^3-2a+1)(k10^3-2a+1) \qquad \#\Omega_{3F} = (10^3-2a+1)^2(k10^3-2a+1))$$

Partimos de la relación $u_t = \Delta u + g(u,t)$ con $g(u,t)$ y $\Delta u$ conocidos, y de la condición de Dirichlet en el dominio discretizado presentado, es decir, conocemos la temperatura $u(x,t)$ para todo tiempo $t \in T = [0,1]$ con $x \in \partial D$) y además para tiempo $t = 0$ conocemos la temperatura en todo el dominio, $u(x,0)$, $x \in D$. El sistema se representará de una forma matricial de la forma en la que se ha descrito en la sección 4, aunque no será estrictamente necesario a la hora de programarlo. En este caso los elementos de la frontera serán los que tengan alguna de sus coordenadas $0 \pm a, N_x \pm a, N_y \pm a, N_z \pm a$.Esto se traduce a que su matriz estará formada por bloques de matrices tridiagonales y $-Id$ de diferentes dimensiones, que vendrá dado según si lo vecinos de un punto son de la frontera o no. El número de incógnitas será menor que en los casos presentados en las secciones anteriores, y no solo eso sino que también el orden de ellas representadas por el vector $b_{front}$ y la matriz laplaciana resultante será diferente si queremos separar los términos que vayan a representar más adelante nuestras incógnitas del sistema lineal. En cuanto al código es de una dificultad similar ya que utilizando condicionales 'if' sobre los índices de las coordenadas espaciales se soluciona el problema. Por ejemplo en el caso tres-dimensional y siguiendo la notación presentada con aterioridad, siendo $i \in \{1, ..., N_z - 1\}$, $j \in \{1, ..., N_y - 1\}$, $h \in \{1, ..., N_x - 1\}$ y $l$ dimension del vector $b_{Front}$

```
if((i>=a && i<=k*Nz-a)&&(h>=a && h<=Ny-a)&&(j>=a && j<=Nx-a))
    l--;
else
  //proceso
```

Después de este proceso, se tendrá claro que puntos son incógnitas, representados en $AU_{t+\delta}$ , y cuales son conocidos (puntos frontera), $b$. Por lo tanto se tiene el sistema $AU_{t+\delta} = b$ que hay que solucionar.

## 6.2. Sistemas Lineales

Para solucionar el sistema lineal que se plantea, se utilizarán únicamente los métodos de S.O.R. y Gradiente conjugado para plasmar las diferencias de los métodos lineales. Técnicamente en el método de S.O.R, cuando se quiere calcular el valor de un elemento de $U_{t+\delta}$, hay que tener en cuenta que elementos vecinos son frontera para sumarlos al vector $b_{Front}$ por que no son incógnitas. A la hora de programarlo será necesario, para calcular el sumatorio del método, acumular la suma de lo términos de los vecinos que no son frontera. Por lo tanto habrá el doble de condiciones que con el dominio más simple, ya que ahora tenemos 12 planos como frontera, mientras que antes teníamos la mitad. En Gradiente Conjugado, una vez que ya se tienen los coeficientes de la matriz del sistema y el vector $b$, el programa es exactamente el mismo, es decir, no habrá que añadir más condicionales al método, aunque sí, en la multiplicación de la matriz $A$ y la definición del vector $b$.

## 6.3. Visualización de los resultados y análisis

En este caso se utilizará la misma función que en el aparatado 5.4 $u(x, y, z, t) = cos(x + y + z + t)$ y se le dará ese valor a lo largo de la frontera de nuestro dominio. Compararemos los resultados aproximados con la solución real y cogeremos la máxima diferencia calculada, a partir de los cada punto del dominio espacio-temporal.

| | | nº iteraciones | tiempo de ejecución | Error máximo |
|---|---|---|---|---|
| Jacobi | Implícito | 2294 | 180s | $2.897\times10^{-5}$ |
| | Crank-Nicolson | 1784 | 140s | $1.135\times10^{-7}$ |
| Gauss-Seidel | Implícito | 1197 | 90s | $2.853\times10^{-5}$ |
| | Crank-Nicolson | 932 | 70s | $1.135\times10^{-7}$ |
| SOR | Implícito | 161 | 12s | $2.865\times10^{-5}$ |
| | Crank-Nicolson | 132 | 10s | $1.138\times10^{-7}$ |
| Gradiente Conjugado | Implícito | 118 | 6s | $2.865\times10^{-5}$ |
| | Crank-Nicolson | 99 | 4s | $1.179\times10^{-7}$ |

| | | nº iteraciones | tiempo de ejecución | Error máximo |
|---|---|---|---|---|
| Jacobi | Implícito | 3458 | 2492 | $2.798\times10^{-5}$ |
| | Crank-Nicolson | 2672 | 1748 | $1.772\times10^{-7}$ |
| Gauss-Seidel | Implícito | 1802 | 350m | $3.252\times10^{-5}$ |
| | Crank-Nicolson | 1394 | 275m | $1.772\times10^{-7}$ |
| SOR | Implícito | 215 | 40m | $3.252\times10^{-5}$ |
| | Crank-Nicolson | 173 | 35m | $1.771\times10^{-7}$ |
| Gradiente Conjugado | Implícito | 157 | 21s | $3.252\times10^{-5}$ |
| | Crank-Nicolson | 134 | 19s | $1.735\times10^{-7}$ |

Se puede ver en la tabla de resultados cómo a medida que bajamos, una vez más, el error disminuye, por lo que se acerca más la solución, y el número de iteraciones y tiempo de ejecución es considerablemente menor. Como también es de esperar se nota la diferencia entre la aproximación a la derivada $u_t$ de segundo orden de Crank-Nicolson y con la de Euler implícito de primer orden. También se puede observar como el error es similar al del dominio anterior y no cambia según el método de resolución lineal. Cuando en SOR se computa el error $w \in (0, 2)$ con un paso de 0,05, el que se ha considerado como $w$ óptimo está en los valores $1{,}75, 1{,}8, 1{,}85$ dependiendo del caso en del método y de la dimensión.

Los resultados se han guardado pero no se han visualizado en dimensión tres por el alcance de este trabajo, la representación de los resultados de la aproximación de la función $\cos(x, y, z, t)$ serán los mismos que los representados en el ejemplo del cuadrado, pero sin el cuadrado más pequeño del interior (dejando un ancho de $a$ entre las fronteras), por lo que no se mostrará gráficamente aunque se puede conseguir utilizando una vez más el software de *g*nuplot . Otra herramienta que se puede utilizar es *fmpeg*, que ayudará en producir un vídeo con las soluciones

que tengamos utilizando *gnuplot* plasmadas en un archivo con extensiones gráficas (como '.png' en nuestro caso). Con esto se podrá interpretar mejor el resultado y con más rapidez.

# 7. Posibles líneas de avance

Después de ver una posible solución para encontrar la temperatura de dominios cúbicos dada la relación diferencial que define Fourier, hay que tener en cuenta las posibles mejoras o los posibles planteamientos del problema. El plantarse este problema en según que dominios diferenciables, hace que se pierda precisión al discretizar el dominio. En estos casos el método de elementos finitos pueda dar una mejor aproximación ya que la discretización se basa en una triangulación del dominio más realista. Por otro lado los propios pasos que se han planteado a lo largo del trabajo pueden ser mejorados. A continuación se presentará una posible manera de alterar el método lineal Gradiente Conjugado y mejorar sus propiedades. Otra posible linea de trabajo es encontrar la manera de interpretar el resultado de cara a sacar conclusiones con ayuda de la visualización de dominios de tres dimensiones en un plano (p.e. la pantalla) que presenta algún reto.

## 7.1. Precondicionando el método de Gradiente Conjugado

Partimos con un sistema lineal $Ax = b$ que ha sido solucionado con el método de Gradiente Conjugado explicado anteriormente y queremos, sin alterar la lógica del método, aplicar un precondicionamento al sistema que consiste en multiplicar los dos lados de la igualdad por una matriz $M$ con la intención de mejorar las propiedades de la matriz multiplicando ambos lados de la igualdad por una matriz. Como la convergencia de la iteración de una matriz depende de las propiedades de la matriz del propio sistema( valores singulares entre otros), se precondiciona esta para que cambiar sus propiedades para que el método converja con mayor rapidez. En lugar de depender de la matriz $A$, si utilizamos la igualdad, la convergencia dependerá de las propiedades de $M^{-1}A$ en su lugar. Si está bien escogida podremos dotar al sistema de las propiedades que precisamos, como puede ser tiempo de cálculo aproximación de la condición inicial a la solución.

El cómputo del producto de matrices $M^{-1}A$ hay que plantearlo de una manera eficiente ya que es el producto de matrices de grandes dimensiones. Como el sistema que se obtiene del método de diferencias finitas es una matriz *sparse* (la mayoría de coeficientes nulos, menos la diagonal), a niveles prácticos no se computará explícitamente la matrix $M^{-1}A$ ni $M^{-1}$, en cambio, se construirá la solución del sistema de ecuaciones de la forma $M^{-1}A$ y $My = c$. Se reescribe el sistema considerando

$$Ax = b \iff E^{-1}A((E^T)^{-1}E^T)x = E^{-1}b$$

y definiendo

$$\tilde{A} = E^{-1}AE^{-T}, \quad \tilde{b} = E^{-1}b \quad y = E^T x$$

para encontrarnos con un sistema de la forma $k$ que se resolverá planteándolo como un sistema de minimización de la siguiente forma:

$$\min_{y\in\mathbb{R}^n} \frac{1}{2}y^T\tilde{A}y - y^T\tilde{b} + c \tag{7.1}$$

y se utiliza minimizar este sistema por el hecho de que ahora podemos utilizar la matriz $E$ de tal manera que $\kappa(\tilde{A}) << \kappa(A) = \frac{\lambda_{\text{máx}}(A)}{\lambda_{\text{mín}}(A)}$ o que agrupe el espectro de $\tilde{A}$ en pocos valores positivos para mayor convergencia del mínimo. Si definimos $M = EE^T$ entonces tenemos una matriz *sparse* que satisface:

$$E^{-T}\tilde{A}E^T = E^{-T}E^{-1}AE^{-T}E^T = E^{-T}E^{-1}A = (EE^T)^{-1}A = M^{-1}A$$

por lo tanto las matrices $\tilde{A}$ y $M^{-1}A$ tienen los mismos valores propios y el sistema (7.1) queda determinado por la matriz $M$.

Casos simples que podemos considerar para hacernos una idea del método, son los casos $M = A$ y $M = I$. En el primer caso la resolución será simple, ya que tenemos el sistema $x = A^{-1}b$ y únicamente hay multiplicar la matriz $M^{-1}$ por el vector $b$, pero la problemática viene en poder calcular la inversa de una matriz de tales dimensiones, por lo tanto nada se gana. En el segundo caso aplicar el precondicionador es fácil (ya que es la identidad) pero el sistema precondicionado es el mismo que el sistema original. Entre estos dos casos extremos existen los sistemas útiles de precondicionamiento tal que $My = c$ se resuelva rápidamente y cercana a $A$ en el sentido de que converja mejor el método iterativo para $M^{-1}Ax = M^{-1}b$. La dificultad de precondicionar la matriz viene por encontrar una que nos ayude a mejorar la convergencia del sistema. No se conoce una forma metódica de encontrar una matriz óptima, no obstante, existen varios procesos para encontrar una.

De forma natural es problema nos sugiere una pregunta. ¿Qué quiere decir, suficientemente cercano a $A$? Si los vaps de $M^{-1}A$ son cercano a 1 y la norma $||M^{-1}A - I||_2$ es pequeña, entonces cualquier iteración se puede considerar que vaya a converger rápidamente. Aún así hay precondicionadores que no cumplen esta condición tan fuerte y en cambio funcionan bien. Por ejemplo, los valores propios de $M^{-1}$ podrían estar cerca de otro valor que no fuera 1, o podría haber algún vap alejado de los demás. Los detalles de esta pregunta de ratio de convergencia depende, como siempre, en problemas de aproximación de polinomios en el plano complejo. Todo lo que cambia para el análisis de precondicionadores en contraste con las iteraciones básicas es que ahora son las propiedades de $M^{-1}A$ las que nos interesan en lugar de las de $A$. Utilizando esta lógica de que la matriz $M$ que precondiciona el sistema tiene que estar cerca de la matriz $A$ del sistema es coherente utilizar las matrices que salen de un método iterativo, como por ejemplo el método de SOR. También se aplican lógicas como a los métodos que NO son GCN(Gradiente Conjugado para ecuaciones normales) donde se asume que un precondicionador es bueno si $M^{-1}A$ no está alejado de las normales y sus valores propios están cerca (aglomerados).

A nivel de programación no es difícil modificar el método del Gradiente conjugado para este sistema ya que se resolverá el sistema $\tilde{A}y = \tilde{b}$.Por lo tanto, siguiendo la misma lógica que antes, tendrá el siguiente pseudo-código:

$$
\begin{aligned}
&Inicializacion: \\
&\qquad r_0 = p_0 = b - Ax_0; \\
&\qquad z_0 = M^{-1} r_0 \\
&\qquad k = 0; \\
&while \quad true: \\
&\qquad \alpha_k = \frac{r_k^T z_k}{p_k^T A p_k} \\
&\qquad x_{k+1} = x_k + \alpha_k p_k \\
&\qquad r_{k+1} = r_k - \alpha_k A p_k \\
&if \quad r_{k+1} < tolerancia \quad break; \\
&\qquad z_{k+1} = M^{-1} r_{k+1} \\
&\qquad \beta_k = \frac{z_{k+1}^T r_{k+1}}{z_k^T r_k} \\
&\qquad p_{k+1} = z_{k+1} + \beta_k p_k \\
&\qquad k = k + 1
\end{aligned}
$$

## 7.2. Visualización 3D

Otra posible manera de avanzar en este proyecto sería conseguir la posibilidad de visualizar los resultados que se recogen para tres dimensión. Para representar estos resultados se tienen que representar 5 dimensiones, las tres espaciales , la temporal y la temperatura con un degradado de color que se entienda la diferencia entre los puntos mas calientes y los puntos más fríos. Para ello hay varias posibilidades, la más complicada sería representarlo en un espacio de realidad virtual donde para ver por capas siendo cada capa un cubo más pequeño que el anterior hasta llegar a la frontera. Otra posibilidad sería con la ayuda de la herramienta *plotly* que nos permite con el lenguaje de programación javascript hacer gráficos con transiciones y por capas, siguiendo el mismo concepto que antes. Con esta herramienta se podría crear presentaciones en el un entorno *LateX*, lo que nos permitiría exponer la visualización más clara y vectorizada de nuestro resultado.

# 8. Conclusiones

El método de diferencias finitas plantea encontrar la temperatura de un dominio a lo largo del tiempo conociendo conociendo la temperatura en tiempo inicial y la de la frontera en todo tiempo. Programando este método y utilizando alguna herramienta de visualización, se ha desarrollado un proceso para aproximar el calor del interior del dominio. Esto ayuda a entender ciertos comportamientos por ejemplo, cuando se tiene una situación de equilibrio y la temperatura no cambia respecto al tiempo, se tiene que cumple el principio del máximo y se sabe que el máximo o mínimo está en la frontera. Otra observación que se puede hacer es, en la relación diferencial, la función $g(u,t)$ se puede interpretar, de manera intuitiva, cómo un cambio de temperatura en el que afecta a todo punto del dominio (cómo un microondas); en el caso que la función $g()$ sea constante se puede ver en la representación gráfica la típica transmisión de calor que sería la que proporciona la frontera. Las aproximaciones que se consiguen con el método son realmente cercanas a la realidad. Como se ha visto en las tablas de resultados, el mínimo error de $\approx 1 \times 10^{-7}$, ha sido calculado utilizando el método de Crank-Nicolson.

Si se quiere hacer con todas las combinaciones de métodos diferenciales y de resolución de sistemas lineales, hay que tener claras las dificultades que se pueden dar. Una de ellas puede ser la discretización del dominio para que se guarde como una matriz, y poderle aplicar los métodos lineales. También puede ser confuso utilizar la lógica condicional en la programación para determinar si los términos son incógnitas o no, que es utilizado tanto en los métodos como en la definición de del $b$ como la multiplicación por $A$. Es muy importante tener recursos informáticos para encontrar solución a los sistemas de visualización, si no se tiene un conocimiento previo, por que el tiempo de búsqueda se reduce considerablemente.

Es importante saber hacia donde se quiere ir una vez que se acaba un proceso como este. Una de las posibilidades que se a planteado es mejorar el método de Gradiente conjugado, para ganar tiempo de ejecución y ser capaz de tratar sistemas más grandes en el mismo tiempo. También se ha nombrado otro problema muy parecido que se plantean con las condiciones de entorno de Neumann, done puede utilizar la misma lógica para solucionarlo. Para visualizar mejor los resultados y poder ver la temperatura de las tres dimensiones por capas a lo largo del tiempo, se han nombrado unas posibles herramientas donde se puede jugar y tener mejor intuición sobre el problema. Aún así, hay posibles mejoras más haya del método de diferencias finitas. Cuando se tratan dominios complicados, donde los puntos más cercanos por cierta dirección no quedan tan claros, se pierde precisión y se utilizan otros métodos que dan mejor resultado. Una de las posibilidades es triangulizar el dominio para que se acerque más a la forma del dominio complejo (los más cotidianos o realistas). Esto se plantea en el método de Elementos Finitos, un proceso que se escapa de este trabajo y que puede ser estudiado con posterioridad a entender este.

Se puede sacar mucho de un ejercicio como este que se presenta. Si se esta en tercer curso de la carrera de matemáticas y se han estudiado los métodos básicos, puede ser una buena manera de utilizar todo lo aprendido y asimilarlo en un ejemplo donde se tendrán que juntar las diferentes piezas para tener un resultado

comprobable y verificar el proceso. Es algo que factible para un alumno con estas características e incluso a más de uno se le puede ocurrir solo con plantearles el problema. El nivel de programación se mejora al programar un proceso con tantos casos y pasos como los que se han presentado y con dimensiones tan grandes. Acaba siendo una buena manera de ver lo que se ha aprendido en 'producción' y conocer los problemas que puede significar eso. A los que le interesé modelar situaciones que se puedan dar en la vida real puede tener un primer contacto en tener la satisfacción que por uno solo se haya aproximado algo que a priori no se conocía con tanta precisión. Redactar todos los pasos y resultados es bueno para asimilar formalmente todos los conocimientos adquiridos y ver como el flujo del trabajo que se ha hecho encaja.

# Referencias


[1] Lloyd N. Trefethen; David Bau (1997) *Numerical Linear Algebra*, Siam.

[2] Apuntes de la asignatura Métodos Numéricos 2 impartida en el Grado de Matemáticas de la Universitat de Barcelona.

[3] Fernando Revilla. 2017. Teorema de los círculos de Gershgorin. [ONLINE] Acesible en: http://fernandorevilla.es/blog/2017/01/04/teorema-de-los-circulos-de-gershgorin/. [Utilizado 15 May 2017].

[4] Sajo Castelli,Andrés M. 2011. [ONLINE] Acesible en: http://159.90.80.55/tesis/000153513.pdf [Utilizado 22 May 2017]

[5] Apuntes UPC - Laboratori de càlcul numèric [ONLINE] Acesible en: https://portal.camins.upc.edu/materials guia/250133/2012/SistemasLineales.pdf [Utilizado 11 Jun 2017]

[6] https://plot.ly

[7] https://trac.ffmpeg.org/wiki/Slideshow

[8] Gene H.Golub, Charles F. Van Loan (2013). Symmetric Eigenvalue Problems *Math Computations, 4th edition* (pp.476-485).The Johns Hopkins University Press Baltimore Maryland.

# Anexos

Anexo I: A continuación el código de los métodos empleados en la figura del horno de tres dimensiones. Se presenta únicamente para reflejar todos los métodos que se utilizan a lo largo del proceso ya que de una forma u otra todos están contenidos en este y es el más complicaciones puede tener. Así se conseguirá un reflejo de lo explicado a lo largo del trabajo con las lógicas de condicionales que se ha comentado en el proceso. Utilizando este código como base, se puede construir el proceso entero en otro archivo principal.

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>

double fronterBi(int i, int j,int k, double ***x, int n,int ancho ,
    int nz);
double  g(double t, double x, double y,double z);
double u0(double t ,double x, double y,double z);
void multMatrizBanda(double *x,int n,double *res, double *a,int
   ancho, int nz);
// funciones basicas
double *** safe3dMatrix(int ,int );
double *** free3dMatrix(double *** ,int ,int );
double * safeVector(int dim);

int metodoGradiente(double ***ut,double *b,int n, double *a,int
   ancho,int nz){
    int k, i,j,l,m=0,n3;
    double  *ap,*rk,*pk,rold=0,rnew=0,alpha, tol=1.e-10;
    n3=(nz-1)*(n-1)*(n-1) - (n-2*ancho+1)*(n-2*ancho+1)*(nz-2*ancho
       +1);
    //guardamos vectores
    ap=safeVector(n3);
    rk=safeVector(n3);
    pk=safeVector(n3);


    for(i=1,l=0;i<nz;i++)
        for(j=1;j<n;j++)
            for (k=1; k < n;k++,l++){
                if((i>=ancho && i<=nz-ancho ) && (k>=ancho && k<=n-
                   ancho ) && (j>=ancho && j<=n-ancho ) ) {
                    l--;
                }else{
                    pk[l] = ut[i][j][k] ;
                }
            }
    //ap=A*pk
    multMatrizBanda(pk,n,ap,a,ancho,nz);

    //r0 = b-Ax0
    //p0=r0
    //rold = r0*r0
    for(l=0;l<n3;l++){
```

```
        rk[l] = b[l] - ap[l];
        pk[l] = rk[l];
        rold+= rk[l]*rk[l];
      }

    while(m<n3){
        //A*pk
        multMatrizBanda(pk,n,ap,a,ancho,nz);

        //alfab=rk*rk  / pk*A*pk
        alpha = 0;
        for(i=0;i<n3;i++) alpha+=pk[i]*ap[i];
        alpha= rold / alpha;



        rnew=0;
        for(i=1,l=0;i<nz;i++)
            for(j=1;j<n;j++)
                for(k=1; k < n;k++,l++){
                    if((i>=ancho && i<=nz-ancho ) && (k>=ancho && k
                        <=n-ancho ) && (j>=ancho && j<=n-ancho ) ) {
                        l--;
                    }else{
                        ut[i][j][k]+= alpha*pk[l];              //
                            x_k+1 = xk + alpha*pk
                        rk[l]-= alpha*ap[l];                    //
                            r_k+1 = rk - alpha*A*pk
                        rnew+=rk[l]*rk[l];                      //
                            new = rk+1*rk+1
                    }
                }

        if(rnew < tol) break;

        rold = rnew/rold;

        for(i=0;i<n3;i++) pk[i] = rk[i] + rold*pk[i];

        rold =rnew;
        m++;
    }
    free (ap);
    free (rk);
    free (pk);
    return(m);

}
void multMatrizBanda(double *x,int n,double *res, double *a,int
    ancho,int nz){ //Multiplicar 'vector' ut por A. Reslutado en
    vector res
    int i,j,l,k;
    double ***pk;

    pk= safe3dMatrix(n+1,nz);
```

```
for(i=1,l=0;i<nz;i++)
    for(j=1;j<n;j++)
        for (k=1;k<n;k++,l++){
            if((i>=ancho && i<=nz-ancho ) && (k>=ancho && k<=n-
                ancho ) && (j>=ancho && j<=n-ancho ) ) {
                 l--;
            }else{
                 pk[i][j][k] = x[l];
            }
        }

for(i=1,l=0;i<nz;i++)
    for(j=1;j<n;j++)
        for (k=1;k<n;k++,l++) {
            if((i>=ancho && i<=nz-ancho ) && (k>= ancho && k<=n
                -ancho ) && (j>=ancho && j<=n-ancho ) ) {
                     l--;
            }else{

                 res[l]=a[0]*pk[i][j][k] ;

                 if(k!=1 && (k!=n-ancho+1 ||(k==n-ancho+1 && ((i
                     <ancho||i>nz-ancho)||(j<ancho||j>n-ancho)) )
                       ) )
                      res[l]+=a[1]*pk[i][j][k-1] ;
                 if(k!=n-1 && (k!=ancho-1 ||(k==ancho-1 && ((i<
                     ancho||i>nz-ancho)||(j<ancho||j>n-ancho)) )
                      ) )
                      res[l]+=a[1]*pk[i][j][k+1] ;

                 if( j!=1  && (j!=n-ancho+1 ||(j==n-ancho+1 &&
                     ((i<ancho||i>nz-ancho)||(k<ancho||k>n-ancho)
                     ) )  ))
                      res[l]+= a[2]*pk[i][j-1][k] ;

                 if(j!=n-1 && (j!=ancho-1 ||(j==ancho-1 && ((i<
                     ancho||i>nz-ancho)||(k<ancho||k>n-ancho)) )
                      ))
                      res[l]+= a[2]*pk[i][j+1][k] ;

                 if(i!=1 && (i!=nz-ancho+1 ||(i==nz-ancho+1 &&
                     ((j<ancho||j>n-ancho)||(k<ancho||k>n-ancho))
                      )  ))
                      res[l]+= a[3]*pk[i-1][j][k] ;

                 if(i!=nz-1 && (i!=ancho-1 ||(i==ancho-1 && ((j<
                     ancho||j>n-ancho)||(k<ancho||k>n-ancho)) )
                     ) )
                      res[l]+= a[3]*pk[i+1][j][k] ;

            }
        }

free3dMatrix(pk, n,nz);
```

```
    return;
}

int Jacobi(double ***ut ,double *a,double *b, double dx , double dy
    ,double dz, double dt, int t ,int n , int nz){
    int i,j,k,l,it=0,ancho=20;
    double  ***utant, max,mcos, uij,sum,tol=1.e-12;
    utant = safe3dMatrix(n+1,nz);
    do{
        for(i=0; i<=nz;i++){
              for(j=0; j<n+1; j++){
                      for(k=0; k<n+1; k++){
                              utant[i][j][k]=ut[i][j][k];
                      }
              }
        }
        max=0;
        mcos=0;
        for(i=1, l=0;i<nz;i++){
            for(j=1;j<n;j++){
                for(k=1;k<n;k++,l++){
                    if((i>=ancho && i<=nz-ancho ) && (k>= ancho &&
                       k<=n-ancho ) && (j>=ancho && j<=n-ancho ) )
                       {
                        l--;
                    }else{
                          sum=0;

                        if(k!=1 && (k!=n-ancho+1 ||(k==n-ancho+1 &&
                            ((i<ancho||i>nz-ancho)||(j<ancho||j>n-
                           ancho)) )  ) )
                            sum+= a[1]*utant[i][j][k-1];
                        if(k!=n-1 && (k!=ancho-1 ||(k==ancho-1 &&
                           ((i<ancho||i>nz-ancho)||(j<ancho||j>n-
                           ancho)) )  ) )
                            sum+= a[1]*utant[i][j][k+1];

                        if( j!=1  && (j!=n-ancho+1 ||(j==n-ancho+1
                           && ((i<ancho||i>nz-ancho)||(k<ancho||k>n
                           -ancho)) )  ))
                            sum+= a[2]*utant[i][j-1][k];
                        if(j!=n-1 && (j!=ancho-1 ||(j==ancho-1 &&
                           ((i<ancho||i>nz-ancho)||(k<ancho||k>n-
                           ancho)) )  ))
                            sum+= a[2]*utant[i][j+1][k];

                        if(i!=1 && (i!=nz-ancho+1 ||(i==nz-ancho+1
                           && ((j<ancho||j>n-ancho)||(k<ancho||k>n-
                           ancho)) )  ))
                            sum+= a[3]*utant[i-1][j][k];
                        if(i!=nz-1 && (i!=ancho-1 ||(i==ancho-1 &&
                           ((j<ancho||j>n-ancho)||(k<ancho||k>n-
                           ancho)) )  ) )
                            sum+= a[3]*utant[i+1][j][k];
```

```
                        uij=(b[l]-sum)/a[0];

                        if(fabs(uij-ut[i][j][k]) > max )
                            max = fabs(uij-ut[i][j][k]);
                        if(fabs(uij-u0(1.*dt*t,1.*dx*i,1.*dy*j,1.*
                           dz*k)) > mcos )
                            mcos = fabs(uij-u0(1.*dt*t,1.*dx*i,1.*
                               dy*j,1.*dz*k));

                        ut[i][j][k] = uij;
                    }
                }
            }
        }
        it++;
    }while(max > tol  && it < 20000);
    free3dMatrix(utant,n,nz);
    return(it);
}

int SOR(double w , double ***ut ,double *a,double *b, double dx ,
    double dy,double dz, double dt, int t ,int n , int nz){
    int i,j,k,l,it=0,ancho=20;
    double  max,mcos, uij,sum,tol=1.e-12;
    do{
        max=0;
        mcos=0;
        for(i=1, l=0;i<nz;i++){
            for(j=1;j<n;j++){
                for(k=1;k<n;k++,l++){
                    if((i>=ancho && i<=nz-ancho ) && (k>= ancho &&
                       k<=n-ancho ) && (j>=ancho && j<=n-ancho ) )
                       {
                        l--;
                    }else{
                        sum=a[0]*ut[i][j][k];

                        if(k!=1 && (k!=n-ancho+1 ||(k==n-ancho+1 &&
                            ((i<ancho||i>nz-ancho)||(j<ancho||j>n-
                           ancho)) )  ) )
                            sum+= a[1]*ut[i][j][k-1];
                        if(k!=n-1 && (k!=ancho-1 ||(k==ancho-1 &&
                           ((i<ancho||i>nz-ancho)||(j<ancho||j>n-
                           ancho)) )  ) )
                            sum+= a[1]*ut[i][j][k+1];

                        if( j!=1  && (j!=n-ancho+1 ||(j==n-ancho+1
                           && ((i<ancho||i>nz-ancho)||(k<ancho||k>n
                           -ancho)) )  ))
                            sum+= a[2]*ut[i][j-1][k];
                        if(j!=n-1 && (j!=ancho-1 ||(j==ancho-1 &&
                           ((i<ancho||i>nz-ancho)||(k<ancho||k>n-
                           ancho)) )  ))
                            sum+= a[2]*ut[i][j+1][k];
```

```
                        if(i!=1 && (i!=nz-ancho+1 ||(i==nz-ancho+1
                            && ((j<ancho||j>n-ancho)||(k<ancho||k>n-
                            ancho)) )  ))
                             sum+= a[3]*ut[i-1][j][k];
                        if(i!=nz-1 && (i!=ancho-1 ||(i==ancho-1 &&
                            ((j<ancho||j>n-ancho)||(k<ancho||k>n-
                            ancho)) )  ) )
                             sum+= a[3]*ut[i+1][j][k];

                        uij=ut[i][j][k]-w/a[0]*(sum -b[l]);

                        if(fabs(uij-ut[i][j][k]) > max )
                             max = fabs(uij-ut[i][j][k]);
                        if(fabs(uij-u0(1.*dt*t,1.*dx*i,1.*dy*j,1.*
                            dz*k)) > mcos )
                             mcos = fabs(uij-u0(1.*dt*t,1.*dx*i,1.*
                                 dy*j,1.*dz*k));

                        ut[i][j][k] = uij;
                    }
                }
            }
        }
        it++;
    }while(max > tol  && it < 20000);
    return(it);
}
void calcBi(int edo, double *b, double ***ut, double ***ut1, double
    dx, double dy,double dz,double dt, int n,int t,int nz){
    double ptsfront, du;
    int i,j,k,h,ancho=20;


    // calculamos b[i] si hay edo
    switch(edo){
        case 1:

            break;
        case 2:
            for( i=1,h=0;i<nz;i++)
                for(j=1;j<n;j++){
                    for(k=1;k<n;k++,h++){
                        if((i>=ancho && i<=nz-ancho ) && (k>= ancho
                             && k<=n-ancho ) && (j>=ancho && j<=n-
                            ancho ) ) {
                             h--;
                        }else{
                             ptsfront = fronterBi(i, j,k, ut,n,ancho
                                 ,nz);
                             b[h] = ut1[i][j][k] + dt*g(1.*dt*t,1.*
                                 dx*i,1.*dy*j,dz*k) + 1.*n*ptsfront;
                        }
                    }
                }
            break;
```

```
        case 3:
            for( i=1,h=0;i<nz;i++)
                for(j=1;j<n;j++){
                    for(k=1;k<n;k++,h++){
                        if((i>=ancho && i<=nz-ancho ) && (k>= ancho
                            && k<=n-ancho ) && (j>=ancho && j<=n-
                           ancho ) ) {
                            h--;
                        }else{
                            ptsfront = fronterBi(i,j,k,ut,n,ancho,
                               nz);
                            du= ut1[i-1][j][k] + ut1[i+1][j][k] -
                               2.*ut1[i][j][k];
                            du+=ut1[i][j-1][k] + ut1[i][j+1][k] -
                               2.*ut1[i][j][k];
                            du+=ut1[i][j][k-1] + ut1[i][j][k+1] -
                               2.*ut1[i][j][k];

                            b[h] = ut1[i][j][k] ;
                            b[h] += 0.5*dt*( g( 1.*dt*(t-1), 1.*dx*
                               k , 1.*dy*j , 1.*dz*i) + g(1.*dt*t
                               ,1.*dx*k,1.*dy*j,1.*dz*i) );
                            b[h] += 0.5*n*(du + ptsfront);
                        }
                    }
                }
            break;
    }

    return;
}
double fronterBi(int i, int j,int k, double ***x, int n,int ancho,
   int nz){
    double ret=0;

    if(k==1)    ret+=x[i][j][k-1];
    if(k==n-1)  ret+=x[i][j][k+1];

    if(j==1)    ret+=x[i][j-1][k];
    if(j==n-1)  ret+=x[i][j+1][k];

    if(i==1)  ret+=x[i-1][j][k];
    if(i==nz-1)  ret+=x[i+1][j][k];

    if(k==ancho-1 &&(i>=ancho && i<=nz-ancho )&&(j>=ancho && j<=n-
       ancho) )    ret+=x[i][j][k+1];
    if(k==n-ancho+1 &&(i>=ancho && i<=nz-ancho )&&(j>=ancho && j<=n
       -ancho) )  ret+=x[i][j][k-1];

    if(j==ancho-1 && (i>=ancho && i<=nz-ancho )&&(k>=ancho && k<=n-
       ancho) )    ret+=x[i][j+1][k];
    if(j==n-ancho+1 && (i>=ancho && i<=nz-ancho )&&(k>=ancho && k<=
       n-ancho) )  ret+=x[i][j-1][k];

    if(i==ancho-1 && (k>=ancho && k<=n-ancho )&&(j>=ancho && j<=n-
```

```
        ancho) )  ret+=x[i+1][j][k];
    if(i==nz-ancho+1 &&(k>=ancho && k<= n-ancho )&&(j>=ancho && j<=
        n-ancho) )  ret+=x[i-1][j][k];


    return ret;
}
void frontera(double ***ut, double dt, double dx, double dy, double
    dz, double t,int n,int nz){  //frontera para todo tempo t
    int i,k,j,ancho=20,mult=nz/n;

        for(k=0;k<=n;k++){
            for(j = 0;j<=n;j++){
                    for(i = 0; i <mult; i++){
                        ut[k+i*n][j][0]=u0(1.*dt*t , 0,  1.*dy*j
                               , 1.*(k+i*n)*dz);
                        ut[k+i*n][0][j]=u0(1.*dt*t , 1.*dx*j,  0
                               , 1.*(k+i*n)*dz);
                        ut[k+i*n][j][n]=u0(1.*dt*t , 1.*dx*n,  1.*
                            dy*j    , 1.*(k+i*n)*dz);
                        ut[k+i*n][n][j]=u0(1.*dt*t , 1.*dx*j,  1.*
                            dy*n    , 1.*(k+i*n)*dz);
                    }

                    ut[0][k][j]=u0(1.*dt*t ,    1.*dx*j    ,  1.*dy
                        *k    , 0);
                    ut[nz][k][j]=u0(1.*dt*t ,    1.*dx*j    ,  1.*
                        dy*k    , 1.*nz*dz);

                    if((k>= ancho && k<=n-ancho ) && (j>=ancho && j
                        <=n-ancho ) ){
                        ut[ancho][k][j]=u0(1.*dt*t ,    1.*dx*j
                            ,  1.*dy*k    , 1.*ancho*dz);
                        ut[nz-ancho][k][j]=u0(1.*dt*t ,    1.*dx*j
                               ,  1.*dy*k    , 1.*dz*(nz-ancho));
                    }

                }
        }
        for(k=ancho;k<=nz-ancho;k++){
            for(j=ancho;j<=n-ancho;j++){
                ut[k][j][n-ancho]=u0(1.*dt*t ,    1.*dx*(n-ancho)
                       ,  1.*dy*j    , 1.*k*dz);
                ut[k][j][ancho]=u0(1.*dt*t ,    1.*dx*ancho   ,
                    1.*dy*j    , 1.*k*dz);

                ut[k][n-ancho][j]=u0(1.*dt*t ,    1.*dx*j    ,  1.*
                    dy*(n-ancho)     , 1.*k*dz);
                ut[k][ancho][j]=u0(1.*dt*t ,    1.*dx*j    ,  1.*dy
                    *ancho    , 1.*k*dz);
            }
        }


    return;
```

```
}
void estadoInicial(double*** ut ,double*** ut1,double dx ,double dy
    ,double dz,int n,int nz){
     int i,j,k;
     for(i=0;  i<= nz ;i++){
         for(j=0;  j<= n  ;j++){
                 for(k=0;  k<= n  ;k++){
                     ut1[i][j][k] = u0(0,1.*dx*k,1.*dy*j,1.*dz*i);
                     ut[i][j][k]= ut1[i][j][k];
                 }

         }
     }
     return;
}
```