Universitat de les Illes Balears

TRABAJO FIN DE GRADO

EXPLORATORIO PARA FÍSICA COMPUTACIONAL CON SIMFLOWNY

Sebastiàn Crespí Guimerà

Grado de Física

Facultad de Ciencias

Año Académico 2021-22

EXPLORATORIO PARA FÍSICA COMPUTACIONAL CON SIMFLOWNY

Sebastiàn Crespí Guimerà

Trabajo de Fin de Grado

Facultad de Ciencias

Universidad de las Illes Balears

Año Académico 2021-22

Palabras clave del trabajo:

Simflowny, Método de líneas, PDEs

Nombre Tutor/Tutora del Trabajo Joan Massó Bennàssar

Nombre Tutor/Tutora (si procede)

Se autoriza la Universidad a incluir este trabajo en el Repositorio Institucional para su consulta en acceso abierto y difusión en línea, con fines exclusivamente académicos y de investigación	Autor		Tutor	
	Sí	No	Sí	No
	☒	☐	☒	☐

Resumen

Este trabajo presenta un estudio acerca de la plataforma *Simflowny*, observando su funcionamiento de cara a un usuario nuevo, así como todas las facilidades de que dispone de cara a un usuario no muy experimentado con ciertos aspectos más avanzados de la computación. Para el estudio de la plataforma, se profundizará en el método de líneas, que supone la base de su funcionamiento y un avance en lo que respecta a los conocimientos de Física Computacional. Se analizará el funcionamiento de *Simflowny* para tres casos: la ecuación de advección, la ecuación del calor y la ecuación de Laplace, siendo el primer caso unidimensional y que funciona a modo de un sencillo ejemplo para introducir la visualización de datos desde la plataforma. Para cada esquema, se observa como una disminución del intervalo espacial o temporal conduce a una mayor resolución de la solución obtenida, lo cual demuestra el buen funcionamiento de los esquemas ya implementados en la plataforma, si bien no se ha podido conseguir la implementación de esquemas presentes en Física Computacional.

Aquest treball presenta un estudi sobre la plataforma *Simflowny*, observant el seu funcionament de cara a un usuari nou, així com totes les facilitats de què disposa de cara a un usuari no molt experimentat amb certs aspectes més avançats de la computació. Per a l'estudi de la plataforma, es profunditzarà en el mètode de línees, que suposa la base del seu funcionament i un avanç en el que respecta als coneixements de Física Computacional. S'analitzarà el funcionament de *Simflowny* per a tres casos: l'equació d'advecció, l'equació del calor i l'equació de Laplace, siguent el primer cas unidimensional i que funciona a mode d'un senzill exemple per introduïr la visualització de dades des de la plataforma. Per a cada esquema, s'observa com una disminución de l'interval espacial o temporal conldueix a una major resolució de la solució obtenguda, el que demostra el bon funcionament dels esquemes ja implementats en la plataforma, si bé no s'han pogut aconseguir l'implementació d'esquemes presents en Física Computacional.

In this project we study *Simflowny*'s platform and its performance from a newcomer's point of view, including all the facilities it has for a user who might lack experience in certain advanced computational aspects. Focusing on the platform's observations, the method of lines will be delved into, which is the basis of its operation and an advance in regards to the knowledge of Computational Physics. Simflowny will be analyzed for three cases: the advection equation, the heat equation and the Laplace equation, being the first case a simple example to introduce data visualization from the platform. For each scheme, it is observed how a decrease in the spatial or temporal interval leads to a higher resolution af the solution obtained, which demonstrates the good functioning of the schemes already implemented in the platform, althought it has not been posible to achieve the implementeation of schemes present in Computational Physics.

Índice

1. Introducción teórica y objetivos

1.1. Introducción teórica: EDPs y diferencias finitas

Las Ecuaciones en Derivadas Parciales (también llamadas EDPs o PDEs por sus siglas en inglés) son un tipo concreto de ecuaciones diferenciales, en el cual encontramos, además de una función u a resolver que depende de como mínimo 2 variables independientes, como mínimo una derivada parcial que dependa de cualquiera de esas variables. Estas suponen una de las bases más importantes de la Física, y su resolución está ligada a multitud de campos: dinámica de fluidos, electromagnetismo, astrofísica...

Las EDPs toman la siguiente forma:

$$F(x_1, x_2, ..., x_n, u, \frac{\partial u}{\partial x_1}, \frac{\partial u}{\partial x_2}, ..., \frac{\partial u}{\partial x_n}, \frac{\partial^2 u}{\partial x_n \partial x_n}) = 0$$

Donde n es un índice que indica a qué variable independiente nos estamos refiriendo. Si, además, F es una función lineal de u, entonces nos encontramos ante una PDE lineal.

Las EDPs de segundo orden, que serán de gran interés a lo largo de este trabajo, se clasifican en tresf tipos importantes. Estos tipos se obtienen a partir de la forma general de una ecuación de segundo orden:

$$Au_{xx} + 2Bu_{xy} + Cu_{yy} + Du_x + Eu_y + F = 0$$

A partir de esta ecuación acudimos a la siguiente matriz **Z**:

$$\begin{bmatrix} B & A \\ C & B \end{bmatrix}$$

El tipo de la EDP de segundo orden depende del determinante:

- Si det(Z) > 0, entonces la EDP es hiperbólica, como la Ecuación de la onda.
- Si det(Z) $= 0$, entonces la EDP es parabólica, como la Ecuación de difusión.
- Si det(Z) < 0, entonces la EDP es elíptica, como la Ecuación de Laplace.

Sin embargo, obviamente la resolución de las ecuaciones presentadas hasta ahora no conduce a una solución única: esta dependería de varias constantes. Se requiere, entonces, de la adición de condiciones de contorno que nos ayuden a esclarecer las condiciones específicas bajo las que se encuentra el problema. Podríamos recurrir, por ejemplo, a condiciones de contorno de Dirichlet (donde se especificaría el valor de la función en la frontera), a condiciones de contorno de Neumann (en este caso hablaríamos del valor concreto de la derivada parcial en el contorno) o sencillamente a condiciones iniciales (necesarias en el caso de ecuaciones que contengan como mínimo una derivada temporal y que indican el estado inicial del problema).

A pesar de que la forma más obvia y directa de afrontar estas problemas es desde el punto de vista analítico, muchas veces no resulta la opción más eficiente. Por ello, la Física Computacional surgió como una alternativa que permitiría la resolución de problemas que, tal vez, no fueran siquiera resolubles de forma analítica.

Estas soluciones serían obtenidas a través de los métodos numéricos, gracias a los cuales pasaríamos de disponer un problema de ecuaciones diferenciales a un problema meramente algebraico, si bien de dimensiones generalmente inmensas si lo que queremos es aspirar a un grado de precisión alto y que, por lo tanto, requiere del desarrollo de algoritmos que permitan solucionar el problema planteado. Dicho de otro modo, es la transición de variables espaciales y temporales desde formas continuas a formas discretas que dependen de mallas.

La forma más simple de trabajar con estas mallas es asumir un incremento espacial o temporal constante a lo largo de todo el dominio. Por ejemplo, vamos a discretizar la ecuación $u_t = u_x$ a través de un esquema FTCS:

$$\frac{U_n^{k+1} - U_n^k}{\Delta t} = \frac{U_{n+1}^k - U_{n-1}^k}{2\Delta x}$$

Donde k es un índice que hace referencia a la discretización temporal de la malla U y n es un índice que hace referencia a la variable espacial. Este método es muy sencillo, puesto que si asumimos una malla inicial que ya conozcamos (conocida gracias a las condiciones iniciales) podemos resolver explícitamente el valor de U_n^{k+1}. Aun así, el uso de estos esquemas también implica ciertas limitaciones que hay que investigar para cada caso que queramos aplicar: la estabilidad, la convergencia... En definitiva, no podemos usar arbitrariamente cualquier esquema para el problema que deseemos, debe haber una investigación profunda previa que nos permita identificar de cual se trata.

1.2. Objetivos

El objetivo de este trabajo es el análisis de los métodos ya conocidos y trabajados a través de la asignatura de Física Computacional y comprobar el funcionamiento de la plataforma de $Simflowny$, que nos permitiría la resolución de EDPs a través del método de líneas, que no había sido presentado en la asignatura.

Discutiremos tres ecuaciones: la de advección, la del calor y la de Laplace, analizando el funcionamiento de la discretización usada y la validez de las soluciones obtenidas, teniendo en cuenta en todo momento qué puede ofrecer *Simflowny* como plataforma para la resolución de estos poblemas y las facilidades que ofrece más allá de lo útil que resulte para enfrentarnos a estas situaciones (no porque sea lo menos importante, sino porque es crucial que la plataforma ofrezca ventajas que permitan que cualquier estudiante pueda usarla sin ningún tipo de confusión).

2. Situación del estudiante después de haber estudiado Física Computacional

A continuación se van a presentar todos los métodos con los que el estudiante ha trabajado al finalizar la asignatura de Física Computacional.

2.1. Método θ

A la hora de trabajar numéricamente con PDEs, se empieza por la ecuación de difusión del calor:

$$U_{xx} - U_t = 0$$

$$U(x,0) = F(x)$$

Esta ecuación se trabaja a través de los métodos numéricos FTCS, BTCS y Theta.

- El esquema FTCS, siendo una discretización explícita, usa una fórmula de 3 puntos para obtener los valores del siguiente nivel temporal. Toma la siguiente forma:

$$U_j^{n+1} = U_j^n + \nu\left(U_{j+1}^n - 2U_j^n + U_{j-1}^n\right)$$
$$\nu = \frac{\Delta t}{(\Delta x)^2}$$

  La consistencia de este método es incondicional, pero su estabilidad está limitada a valores de $\nu \leq \frac{1}{2}$

- El esquema BTCS, siendo una discretización implícita, requiere del uso de ciertos algoritmos para ser resuelta (en el caso de la asignatura se recurre al algoritmo de Thomas):

$$-\nu U_{j+1}^{n+1} + (1+2\nu)U_j^{n+1} - \nu U_{j-1}^{n+1} = U_j^n$$

  Además de ser inconcicionalmente consistente, al igual que el esquema FTCS, se trata de un esquema incondicionalmente estable, lo que permite que la relación entre Δt y $\left(\Delta x^2\right)$ no se encuentre tan estrechamente confinada.

Los dos métodos anteriores pueden ser generalizados a través del método θ, que permite, a través de un parámetro, el uso de cualquiera de los dos esquemas regulándolo. La forma sería la siguiente:

$$-\theta\nu U_{j+1}^{n+1} + (1+2\theta\nu)U_j^{n+1} - \theta\nu U_{j-1}^{n+1} = \left[1 + (1-\theta)\,\nu\delta_x^2\right] U_j^n$$

Donde $0 \leq \theta \leq 1$, encontrándose el esquema FTCS en el límite $\theta = 0$ y el BTCS en el límite $\theta = 1$. Debido a su carácter implícito requiere también del uso de ciertos algoritmos, por ejemplo el algoritmo de Thomas.

Si $\frac{1}{2} \leq \theta \leq 1$, el método es incondicionalmente estable, y para el resto de valores la condición de estabilidad dependerá de que $\nu \leq \frac{1}{2(1-2\theta)}$, comprobándose de forma bastante sencilla que, para el esquema FTCS ($\theta = 0$), se respeta la condición de estabilidad $\nu \leq \frac{1}{2}$.

Obviamente, dada la información previamente presentada sobre ambos esquemas, es incondicionalmente consiste, pero para $\theta = \frac{1}{2}$ es consistente de segundo orden en Δt, mientras que para el resto de valores es de primer orden.

2.2. Upwind, Lax-Wendroff y Leap-Frog

La siguiente ecuación a resolver es la ecuación de advección lineal, que puede ser resultante del desacomplamiento de una ecuación de segundo orden en dos de primer orden:

$$U_{tt} - c_0^2 U_{xx} = 0 \rightarrow U_t + aU_x = 0$$

Se trata, por lo tanto, de una ecuación hiperbólica.

Inicialmente, se sabe que la solución será de la forma $u(x,t) = u^0(x - at)$, siendo transportada a lo largo de las líneas características, donde su forma será de recta en el plano x-t:

$$x = at + cte$$

En este tipo de ecuaciones hay que prestar muchísima atención a la condición CFL, la cual implica que un algoritmo no puede converger a un punto P (ni ser estable) si su dominio de dependencia no incluye la característica que pasa por P. Dicho de otro modo, el dominio numérico debe incluir al dominio físico:

$$\frac{\Delta x}{\Delta t} \geq a \rightarrow \frac{a\Delta t}{\Delta x} \leq 1 \rightarrow \nu \leq 1$$

A continuación presentaremos cada uno de los 3 métodos:

2.2.1. Upwind

La discretización a usar depende de si $\nu < 0$ o $\nu > 0$, dando lugar a:

$$U_j^{n+1} = \begin{cases} (1 - \nu\Delta_{+x})\, U_j^n, & si \quad \nu < 0 \\ \\ (1 - \nu\Delta_{-x})\, U_j^n, & si \quad \nu > 0 \end{cases}$$

Recordemos que, debido a la condición CFL, $\nu \leq 1$, coincidiendo además con la condición de estabilidad.

A partir del análisis de Fourier a través del modo $u(x,t) = e^{i(k\Delta x + \omega t}$:

$$|\lambda|^2 = 1 - 4\nu(1-\nu) sin^2\left(\frac{k\Delta x}{2}\right)$$

$$arg(\lambda) = -\nu k\Delta x \left[1 - \frac{1}{6}(1-\nu)(1-2\nu)\left(k\Delta x\right)^2 + ...\right]$$

El error de amplitud sería de orden 2, al igual que el error de fase. Sin embargo, si $\nu = 1$, desaparecen tanto el error de fase como el error de amplitud de orden mayor a 1. Si, en cambio, $\nu = 0{,}5$, solo desaparecen las componentes del error de fase de segundo orden, pero no las del error de amplitud.

2.2.2. Lax-Wendroff

Se puede recurrir a la interpolación de órdenes superiores para obtener mejores esquemas. A través de una interpolación de 3 puntos, considerando una recta característica con pendiente a = ct., se obtiene:

$$U_j^{n+1} = \frac{1}{2}\nu(1+\nu)U_{j-1}^n + (1-\nu^2)U_j^n - \frac{1}{2}\nu(1-\nu)U_{j+1}^n$$

A través del análisis de Fourier se sabe que la condición para la estabilidad es que $|\nu| \leq 1$, el rango completo para la condición CFL. Usamos de nuevo el modo $u(x,t) = e^{i(k\Delta x + \omega t}$ se llega a:

$$|\lambda|^2 = 1 - 4\nu(1-\nu)sin^4\left(\frac{k\Delta x}{2}\right)$$

$$arg(\lambda) = -\nu k\Delta x \left[1 - \frac{1}{6}(1-\nu)(1-2\nu)\left(k\Delta x\right)^2 + ...\right]$$

Se observa un mayor amortiguamiento en la amplitud. Sin embargo, ewn este caso el error de amplitud es de orden 4, comparado con el orden 2 que presenta el esquem a Upwind para $(k\Delta x)$. De nuevo, se encuentra un eror de fase de orden 2 en $(k\Delta x)$. Sin embargo, la disminución del error de amplitud compensa el error de fase.

2.2.3. Leap-frog

Este esquema usa una diferencia centrada en el tiempo y en el espacio. Para $a = cte$, toma la forma:

$$U_j^{n+1} = U_j^{n-1} - \nu(U_{j+1}^n - U_{j-1}^n)$$

En este caso, obviamente necesitaremos de otro método para calcular la primera iteración. U^0 vendrá dado por las condiciones iniciales, y U^1 podrá ser calculado a través de, por ejemplo, Lax-Wendroff.

El cumplimiento de la condición de estabilidad implica, al igual que el resto de métodos, el cumplimiento de la condición CFL. Al llevar a cabo el análisis de $\lambda(k)$, lo que conduce a problemas puesto que uno de los valores proporcionará soluciones espúreas.

Tenemos, por lo tanto, una solución real:

$$arg(\lambda_+) \sim -\nu k \Delta x \left[1 - \frac{1}{6}(1-\nu^2)\left(k\Delta x\right)^2 + ...\right]$$

y tomando la raíz negativa se obtiene una solución espúrea oscilante que viaja en dirección contraria,

$$arg(\lambda_-) \sim (-1) \left[1 + i\Delta x + (-\frac{1}{2})\nu^2 k^2 (\Delta x)^2 + ...\right]$$

3. *Simflowny*

Simflowny es una plataforma que genera código de forma eficiente de cara al desarrollo de modelos dinámicos para diferentes marcos de simulación. Puede llegar a resolver problemas de hasta 3 dimensiones y está pensada para plataformas de supercomputación con cálculo paralelo, si bien a lo largo de este trabajo nos centraremos en la faceta pedagógica. Esta discretización puede llevarse a cabo usando:

- Diferencias finitas
- Métodos High-Resolution-Shock-Capturing
- Métodos de partículas

Se trata de una plataforma abierta, compuesta por un Entorno de Desarrollo Integrado (IDE) basado en web y DSL, con una GUI (Graphical User Interface) enfocada en resultar lo más llamativa y sencilla de cara al usuario (tal y como se puede observar en la siguiente figura). Su meta, tal y como se ha indicado previamente, es la generación del código final de simulación.

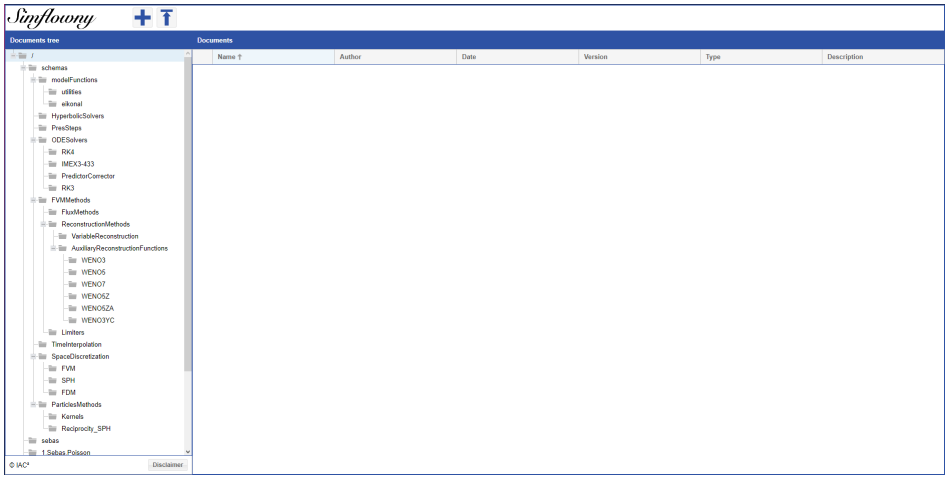


Figura 1: Interfaz de usuario de *Simflowny*, donde podemos observar como se organizan los archivos.

La primera versión introdujo el DSL básico, junto a una famila de PDEs y una primera versión de la GUI, y fue capaz de generar código automáticamente para Cactus y SAMRAI. La segunda versión extendió la familia inicial de PDEs, renovando la GUI y dando soporte a los Agent Based Models con BOOST como marco de referencia. La tercera y última versión genera código para la infraestructura SAMRAI, lo cual no significa que $Simflowny$ no pueda generar código para otras plataformas, pero SAMRAI es el único marco que permite sacarle el máximo provecho a todas las funcionalidades de la última version. Simflowny 3 es de código abierto, y está disponible en la forma de código fuente compilable y también como un container de Docker.

Tal y como veremos en los próximos apartados, la GUI de $Simflowny$ permite la combinación, en un solo problema, de todas las diferentes estrategias de discretización. La combinación de estas políticas de discretización es llevada a cabo tanto en un sentido abstracto como extendiendo de forma práctica la plataforma de simulación de SAMRAI para dar soporte a todos estos métodos de discretización.

El Lenguaje de Domino Específico basado en un Esquema de definición XML. Los esquemas XSD prescriben la estructura de los documentos XML para modelos, problemas y esquemas de discretización.

El DXL soporta un paradigma general para la evolución de una PDE, incluyendo cualquier tipo de derivadas espaciales pero solo derivadas temporales de primer orden en tiempo (algo que no supone una limitación en absoluto dada la naturaleza de las ecuaciones a resolver en los distintos apartados). Como los dos paradigmas más destacados nos encontramos con:

- PDEs escritas en forma de ley de equilibrio. En este caso las PDEs son escritas como un sistema en evolución que solo contiene derivadas de primer orden tanto en tiempo como en espacio. Las derivadas espaciales son siempre representadas como la derivada de algún flujo que depende de los campos, permitiendo el uso de esquemas numéricos basados en Métodos de Volumen Finitos para tratar las discontinuidades. Un ejemplo de ello sería la ecuación de ondas.

- PDEs de evolución, permitiendo las formas más arbitrarias de un sistema en evolución incluyendo derivadas espaciales de cualquier orden, que son discretizadas, con operadores de diferencias finitas estándar. La única restricción es que el sistema debe ser aún de primer orden en tiempo. Esta no es realmente una restricción imperante, puesto que cualquier sistema PDE puede ser reducido a uno de primer orden introduciendo variables adicionales para las derivadas temporales de los campos evolucionados.

$Simflowny$ combina estas 2 familias en una sola, permitiendo cualquier combinación posible en las PDEs, en adición a las formas preexistentes. Además, permite la representación de estas PDEs ya sea en una matriz regular, o en un conjunto de (desestructuradas) partículas en movimiento.

Desde el punto de vista de la ciencia computacional podemos encontrar tres niveles:

- Un nivel de presentación, implementado a través de una IDE basada en un navegador web con una GUI.
- Un nivel lógico, basado en un servidor aplicación.
- Un nivel de datos, combinando bases de datos nativas en XML con almacenamiento masivo de datos.

El proceso de convertir un modelo matemático en un código numérico puede ser dividido en cuatro etapas:

1. La representación del modelo matemático, que contiene las ecuaciones de PDE que van a evolucionar.
2. La representación del problema, que incluye el modelo matemático, el dominio de la simulación, las cantidades de análisis y las condiciones iniciales y de contorno a a plicar a los campos de evolución (en PDEs), así como la condición de finalización (que generalmente añadiremos a través de un límite temporal definido como "tend").
3. La representación del esquema discreto, que convierte el problema continuo en uno discreto definiendo los operadores de espacio y tiempo.
4. La generación del código para el problema discretizado en el marco de simulación. Estos marcos de simulación jugarán el rol de un organizador de matriz/partículas definiendo el dominio, distribuyendo el uso de memoria de los campos y paralelizando la carga de trabajo a través de los distintos procesadores.

$$\partial_t \mathbf{u} + \partial_{\mathbf{i}} \mathbf{F}^{\mathbf{i}}(\mathbf{u}) = \mathbf{S}(\mathbf{u}) + L(\mathbf{u}, \partial_{\mathbf{i}} \mathbf{u}) \tag{1}$$

Donde $\mathbf{u}$ es una *array* con todos los campos evolucionados, $\mathbf{F}^{\mathbf{i}}$ es el flujo en la dirección x_i, $\mathbf{S}$ es el término de origen y $L(\mathbf{u}, \partial_{\mathbf{i}} \mathbf{u})$ es cualquier operador dependiente de los campos y sus derivadas espaciales de cualquier orden. Esta forma es ciertamente redundante, puesto que las derivadas de los flujos siempre pueden ser expresadas usando el operador L. De cualquier forma, cuando los flujos son explícitamente identificados, el modelo permite la aplicación de esquemas de discretización HRSC específicos a los flujos para lidiar con posibles discontinuidades y *shocks* apareciendo en la solución.

4. Método de líneas

Hasta ahora se ha asumido, a lo largo del texto, que se iba a aplicar el método de diferencias finitas para la discretización. Sin embargo, uno de los objetivos de este proyecto es, precisamente, trabajar con un método que resulta nuevo teniendo en cuenta los conocimientos adquiridos en Física Computacional: el método de líneas.

La esencia subyacente de este método es muy sencilla, pues solo requiere ir un paso más allá del método de diferencias finitas. Concretamente, solo vamos a sustituir las derivadas espaciales en las PDE con aproximaciones algebraicas. Una vez hecho esto, las derivadas espaciales dejan de depender de las variables espaciales independientes, pues a la hora de discretizar las derivadas espaciales, pasamos de x a la variable Δx, la cual, como ya sabemos a través del método de diferencias finitas, es sencillamente un valor que establecemos nosotros mismos antes de dar inicio a la solución del problema.

De este modo, solo permanece una variable independiente, el tiempo, que sería una variable continua. A través de este planteamiento, ahora disponemos de un sistema de ODEs que representa la PDE original. El reto recaería en formular el sistema aproximado de ODEs, para luego aplicar cualquier algoritmo de integración y computar una solución numérica aproximada a la PDE.

Vamos a tomar un ejemplo sencillo, la Ecuación del calor $u_t = Du_{xx}$. Usaremos el método de diferencias finitas para la derivada espacial, usando una discretización sencilla:

$$u_{xx} \sim \frac{u_{i+1} + u_{i-1} - 2u_i}{(\Delta x)^2}$$

Donde i es un índice que nos indica e la posición de u a lo largo de una malla desarrollada a lo largo de un eje x y Δx es el incremento espacial escogido, que asumimos constante. Dicho esto, asumimos una malla espacial constituida por N puntos, con un valor inicial $i = 1$ que indica el extremo inferior del eje. Por ello, teniendo en cuenta lo que se ha explicado del método de líneas, tenemos un sistema de N ODEs con la misma forma:

$$\frac{du_i}{dt} = D\frac{u_{i+1} + u_{i-1} - 2u_i}{(\Delta x)^2}$$

Hay que subrayar, por lo tanto, que la esencia del método de líneas recae en este procedimiento: adaptar las PDEs de forma que tenemos que resolver un sistema de ODEs. Esto plantea una pregunta: ¿Cómo integramos las ODEs?

De seguir el método de diferencias finitas la respuesta sería muy sencilla: podríamos usar una diferencia finita de primer orden en el tiempo:

$$\frac{du_i}{dt} \sim \frac{u_i^{n+1} - u_i^n}{\Delta t}$$

Donde n es otro índice, en este caso representativo del orden temporal. Esta discretización permite la resolución explícita de la variable u_i^{n+1}, y es la base de varios de los métodos más sencillos que son introducidos en la asignatura. Este sería un caso de esquema FT (Forward in Time). Esta etapa, siguiendo el método de diferencias finitas, implica muchísimo trabajo tanto de programación como de planteamiento: incluso después de haber decidido el uso de un esquema, hay que tener en cuenta factores como la estabilidad (muy importante en el caso de esquemas explícitos, que presentan un mayor número de limitaciones que los esquemas implícitos) y, además, la dificultad que puede implicar el diseño de estos.

Para evitar este tipo de complicaciones, y limitarlas a la elección de un método de discretización temporal adecuado, se suele usar para el método de líneas una serie de rutinas prediseñadas. Con ello, el tosco trabajo de programación de la integración temporal es evitado, y estas rutinas, generalmente ya trabajadas por expertos, cuentan con una robustez y una eficiencia difícilmente igualables por un estudiante.

Todas estas premisas del método requieren, por lo tanto, que se trate de un problema de valores iniciales de Cauchy. Esto, por definición, limitaría la resolución de ecuaciones elípticas, pero su resolución se puede intentar si tenemos en cuenta que, para $t \to \infty$, las condiciones iniciales van cobrando cada vez menos importancia y $u_t \to 0$, con lo que se pueden diseñar problemas independientes del tiempo usando el método de líneas, si bien de forma muy restringida y computacionalmente muy laboriosa por requerir muchas iteraciones y un tiempo muy elevado, tal y como comprobaremos más adelante.

5. Creación de un problema nuevo

Con tal de ejemplificar el funcionamiento de $Simflowny$, se va a ejemplificar la resolución de la ecuación del calor en 2 dimensiones:

$$u_t = K\left(u_{xx} + u_{yy}\right) \tag{2}$$

$$u(x, y, 0) = x^2 + y^2$$
$$u(0, y) = u(100, y) = u(x, 0) = u(x, 100) = 0$$

En el dominio $\Omega = (0, 100) \times (0, 100)$.

5.1. Modelo

Regresamos a la Figura 1. Arriba, a la derecha del logo, logramos encontrar el símbolo +. Si clickamos sobre este, nos aparecen todos los archivos que podemos crear para desarrollar nuevos problemas o cambiar las condiciones de algunos ya planteados:

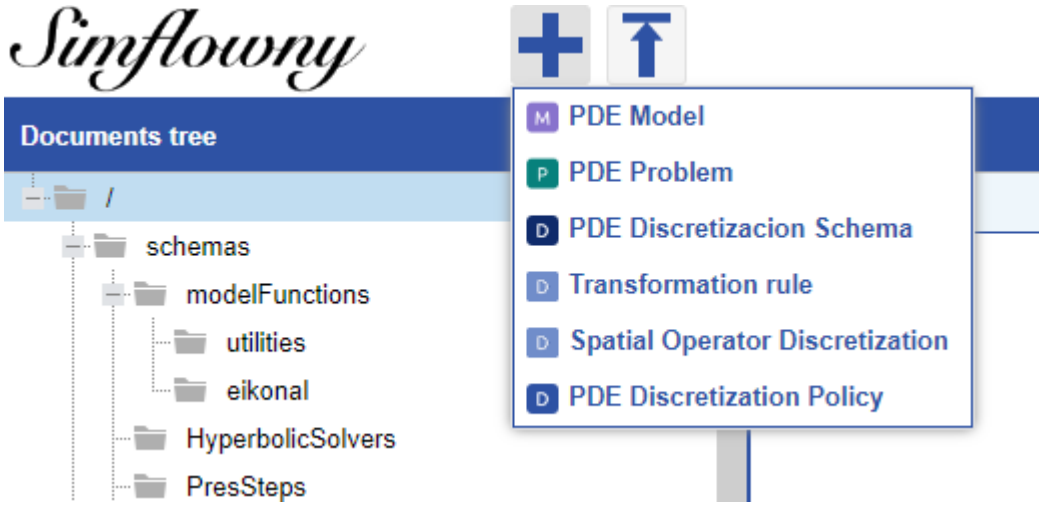


Figura 2: GUI para crear nuevos archivos en la plataforma.

Obviamente, clickamos sobre *PDE Model* y damos inicio al diseño de la ecuación.

Para empezar, empezaremos tratando de plantear el modelo matemático antes de implementarlo en la plataforma. A partir de lo explicado en la Sección previa, se deduce que en la Ecuación 1 tendremos que trabajar a través de operadores $L(\mathbf{u}, \partial_{\mathbf{i}}\mathbf{u})$. Puesto que ambas derivadas son del mismo tipo (derivadas dobles y en una sola dirección), podemos usar el mismo operador para discretizarlas, con lo que vamos a establecer 2 términos dentro del mismo operador. A continuación presentaremos la apariencia de la GUI de *Simflowny* para la creación de modelos matemáticos:

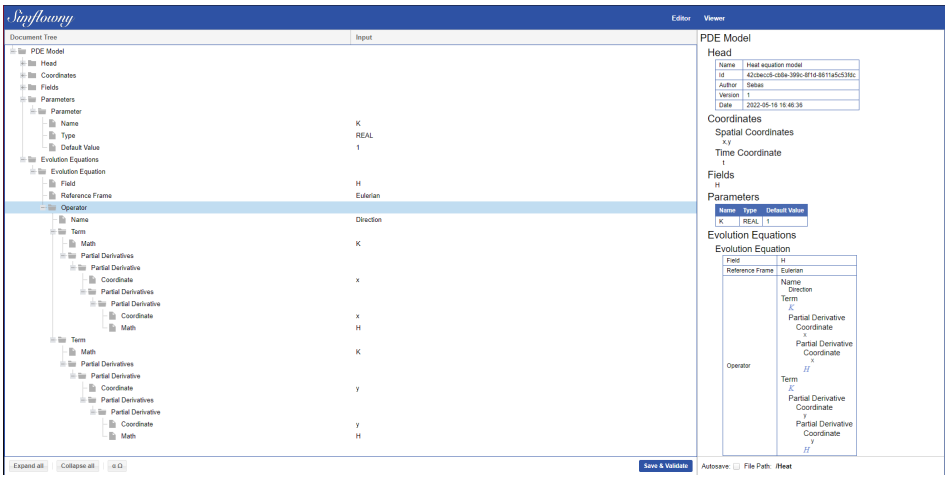


Figura 3: Interfaz de usuario para la creación de modelos de Simflowny.

En esta Figura podemos observar una de las mayores virtudes de la plataforma, que es el ingente número de facilidades que ofrece a la hora de definir cualquier aspecto deseable de la Ecuación. A la izquierda podemos encontrar la sección dedicada al control de las variables a usar en el problema, y a la derecha se puede observar una visualización de estas que permite fácilmente cambiar cualquiera sin navegar a través del árbol de archivos.

En el apartado *Head* encontramos todo lo que define al archivo en sí: el nombre del archivo, su ID (que nos permitirá usar el modelo en cualquier problema que lo requiera, como podremos ver posteriormente), su autor, su versión y la fecha en que fue creado.

En *Coordinates*, *Fields* y *Parameters* empezamos a dar forma al problema. Empezamos definiendo las coordenadas, en este caso al ser un problema en $2 + 1$ dimensiones sencillamente definiremos las coordenadas espaciales x, y y t. Además, hemos definido el campo que vamos a usar como H y el único parámetro a usar, $K = 1$. En este último caso, cabe decir que podemos definir un parámetro INT (como hemos hecho aquí para mayor simplicidad), $REAL$, $BOOLEAN$ o $STRING$.

En *Evolution equations* definimos la Ecuación 2, por ahora haciendo caso omiso a las condiciones de contorno o iniciales. No hay términos de flujo, con lo que bastará definir un operador que englobe los dos términos espaciales de la Ecuación 2, puesto que ambos términos pueden ser discretizados a través del mismo esquema, tal y como veremos posteriormente.

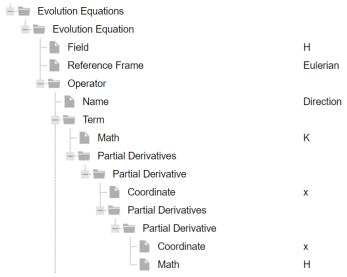


Figura 4: Ampliación de imagen del apartado *Terms*.

La jerarquía define las derivadas y como se aplican. En este caso (ignoramos los términos *Math*, que ya explicaremos más tarde), primero añadimos una rama *Partial Derivatives* a *Term*. Dentro de esta, definimos la coordenada sobre la cual aplicaremos la derivada, en este caso x. Seguidamente, a *Partial Derivative* añadimos otra rama de *Partial Derivatives*, con lo cual llegamos a presentar el modo de hacer una derivada parcial doble aplicado al campo H. Esto último lo sabemos gracias a la jerarquía de los términos *Math*, puesto que el término H que podemos localizar en la Figura 3 debajo de la coordenada x nos indica el campo o función sobre el cual aplicamos la derivada. La primera rama, el término K, al situarse en el mismo nivel que la primera *Partial Derivatives*, denota multiplicación. Es decir, si sustituyéramos el término *Math* por otro *Partial Derivatives* también en dirección x, por ejemplo, que estuviera situado en el mismo nivel que el definido previamente, nos encontraríamos con una multiplicación $u_x u_{xx}$.

Con esto damos por finalizada la fase en la que planteamos el modelo matemático, le damos a *Save & Validate* y podemos empezar con la representación del problema.

5.2. Representación del problema

Regresamos brevemente a la figura 2, donde entre las opciones disponíamos de "PDE Problem". Clickamos y damos inicio al proceso de establecer las condiciones del problema una vez creado el modelo.

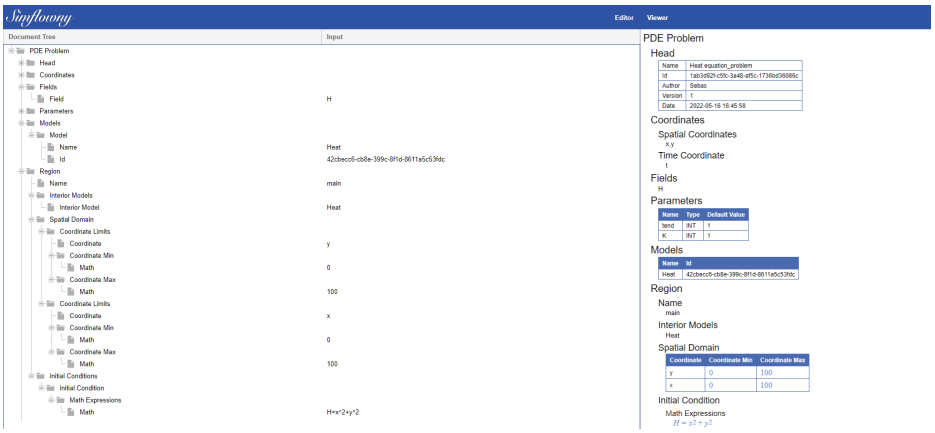


Figura 5: GUI a la hora de desarrollar el problema.

Los apartados *Head*, *Fields*, *Coordinates*, y *Parameters* cumplen las mismas funciones que en el modelo. Cabe notar que, además de las obvias diferencias que encontramos en *Head* debido a que es un archivo diferente, también encontramos un parámetro nuevo llamado *tend*, un INT de valor igual a 1 y cuya utilidad quedará definida más tarde.

Después, en el apartado *Models* importamos a través del ID los modelos (que en este caso solo va a ser uno) que vamos a usar en el problema, que en este caso es únicamente $Heat$. De ser el caso, podríamos definir varias regiones y establecer modelos distintos para cada caso. Un ejemplo de ello sería la ecuación de Schrödinger para un pozo cuadrado, donde tendríamos que resolver una ecuación u otra dependiendo de la zona espacial en que nos encontremos.

En *Region* establecemos los límites espaciales del problema (en este caso el dominio Ω especificado al inicio de la Sección), los modelos interiores a usar (el modelo $Heat$, puesto que no hay más) y las condiciones iniciales. En aras de la simplicidad, hemos escogido como condiciones iniciales $u(x, y, 0) = x^2 + y^2$, tal y como hemos especificado previamente.

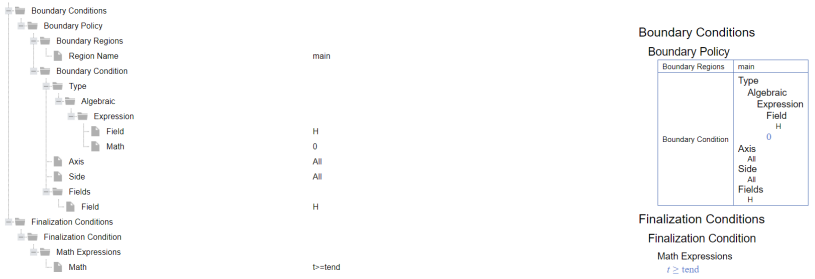


Figura 6: Caption

A continuación, establecemos las condiciones de contorno en *Boundary con-*

ditions, así como las condiciones de finalización de la simulación en *Finalization Conditions*. En *Boundary Policy* escribimos todas las coindiciones de contorno que vamos a usar a lo largo del problema. Podríamos establecer condiciones distintas si hay varias regiones, pero en este caso solo hay una (a la que hemos denominado *main*), así que en *Boundary Regions* introducimos *main*. Ahora nos interesa imponer condiciones de contorno nulas a lo largo de ambos ejes en el campo H, con lo que en *Boundary Condition* introducimos una condición de tipo algebraica en *Type*, introduciendo sencillamente la expresión 0, en *Field* escribimos H y establecemos el cumplimiento de esta condición a lo largo de ambos ejes, en ambos extremos. Finalizamos introduciendo como *Finalization condition* la expresión $t \geq tend$, siendo *tend* el tiempo límite a partir del cual terminará la representación gráfica de la solución, siendo en este caso sencillamente 1 segundo.

5.3. Representación del esquema discreto

5.3.1. *Spatial operator* y *Transformation rule*

Ya hemos designado el aparato puramente matemático para el problema, con lo que procederemos a dedicarnos al apartado puramente computacional, empezando por la discretización del problema. Para la discretización espacial vamos a tener que diseñar un *Spatial Operator*:

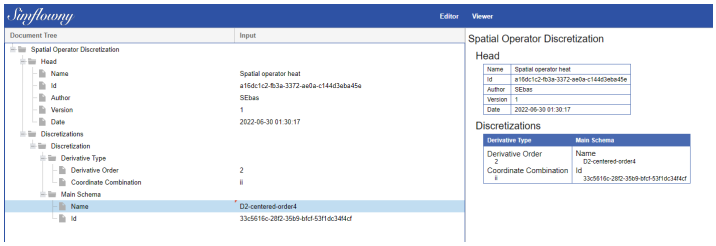


Figura 7: *Spatial operator* diseñado para el problema

En *Discretizations* vamos a definir todas las discretizaciones espaciales que vamos a usar a lo largo del problema. En este caso, al tratarse de la Ecuación del calor, sencillamente tenemos que definir una discretización para una derivada doble de la misma coordenada. En este problema vamos a usar un esquema sencillo centrado de orden 4 (el cual introducimos en *Main Schema* a través de su Id):

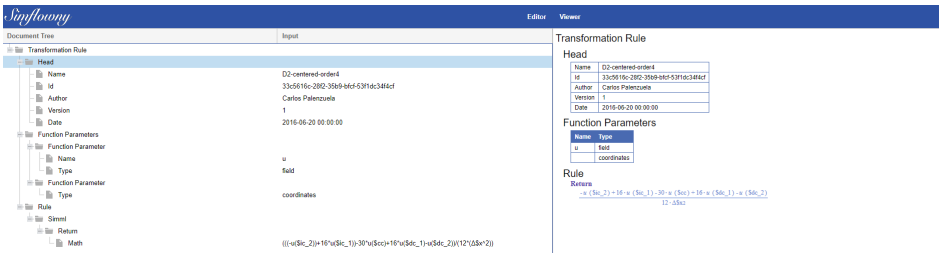


Figura 8: Discretización espacial a usar para las derivadas de orden 2 del problema.

En *Rule* definimos concretamente el esquema. A través de $u(\$cc)$ usamos la celda actual, a través de $u(\$ic\_X)$ definimos el número de celdas respecto al cual avanzamos, y finalmente a través de $u(\$dc\_X)$ definimos respecto a qué número de celdas retrocedemos.Por lo tanto, la discretización presentada en *Rule* se traduce en:

$$U_{i,j}^{n+1} = \frac{-U_{i+2,j}^{n} + 16U_{i+1,j}^{n} - 30U_{i,j}^{n} + 16U_{i-1,j}^{n} - U_{i-2,j}^{n}}{12\Delta x}$$

Así tenemos preparado todo lo que necesitábamos antes de proceder a discretizar el Problema que hemos ido preparando en las Subsecciones anteriores.

5.3.2. *PDE Discretization policy*

Regresamos momentáneamente al instante en que hemos finalizado el diseño del problema. Seleccionamos el archivo del problema y, en la parte superior, nos saldrán varias opciones:

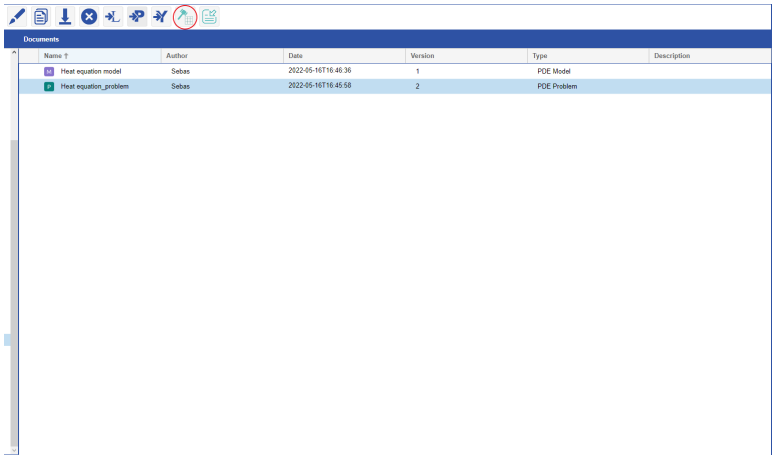


Figura 9: Paso posterior a la creación del problema.

Al clicar sobre el icono indicado surge en el GUI el archivo correspondiente a la discretización:

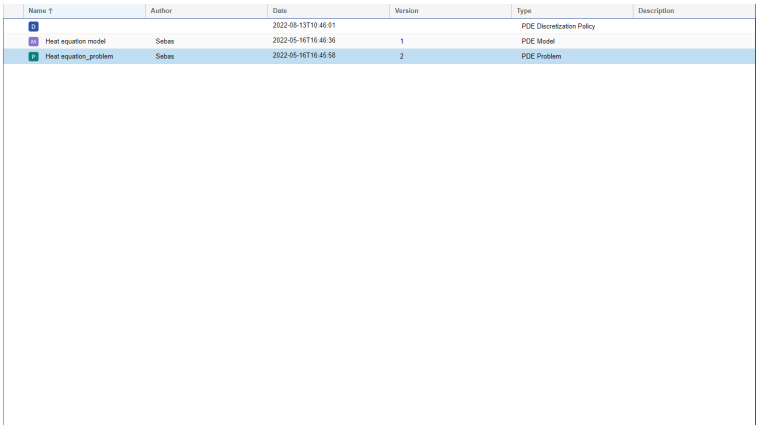


Figura 10: Aparece el archivo correspondiente a la discretización.

Como podremos ver en la siguiente Figura, empezamos a incluír los datos necesarios para la discretización del problema. Después de haber realizado el paso anterior, nos encontramos con que los apartados *Problem* y *Mesh Variables* estan automáticamente rellenados con los datos correspondientes al problema a partir del cual está creado el esquema. El apartado *Head* lo rellenamos de nuevo incluyendo los datos correspondientes al archivo en sí.

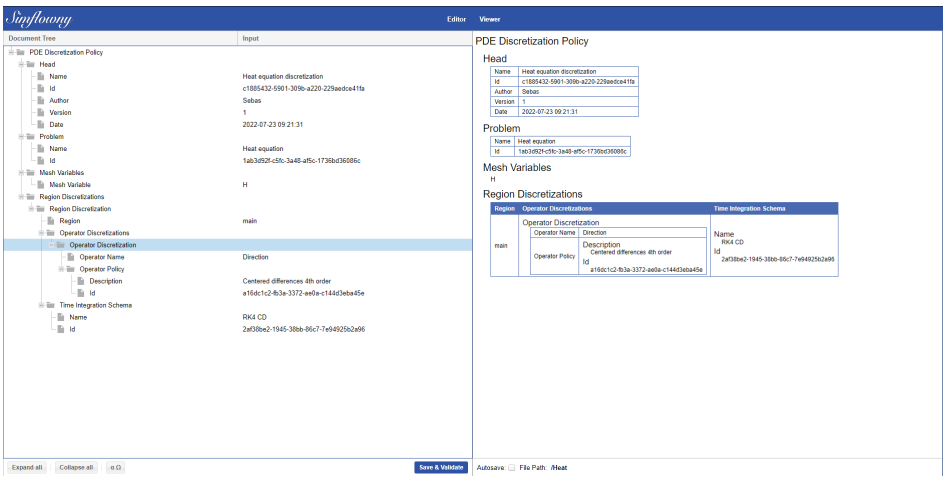


Figura 11: Editor del esquema de discretización.

Es en este paso donde vemos finalmente la utilidad de que las dos derivadas parciales del modelo (la derivada doble respecto a x y la derivada doble

respecto a y), puesto que en ambos casos es obvio que podemos recurrir al mismo esquema de discretización espacial. En este caso, usamos el operaldor que hemos preparado en la subsección anterior, y, a la hora de definir el *Time Integration Schema* usamos uno que nos proporciona la plataforma, el *RK4 CD*. Seleccionamos *Save* & *Validate* y procedemos al paso final.

5.4. Generación del código

Ha llegado el paso final en lo que respecta a la creación del problema. Nos encontramos con 4 archivos en la GUI, tal y como vemos en la figura 12.1. Seleccionamos el esquema discretizado,así como el correspondiente icono de "Discretizar"(situado en la parte superior de la pantalla), y nos encontramos con un nuevo archivo: el problema discretizado.

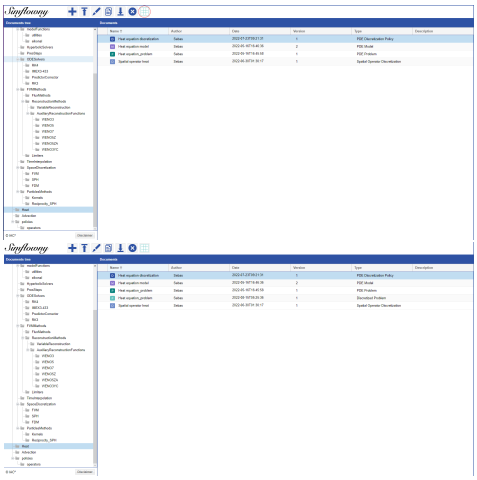
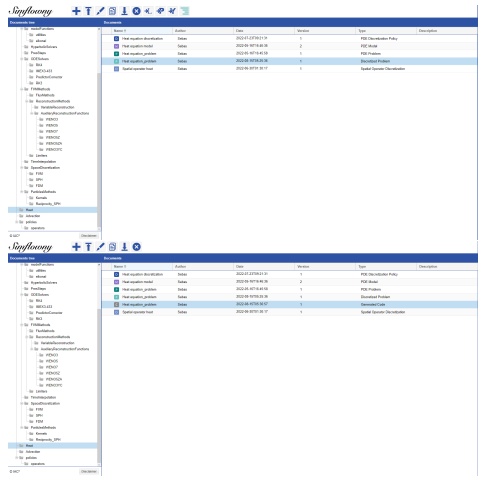


Figura 12: Generación del código a partir del esquema de discretización.

Ahora debemos seleccionar el problema a discretizar y clicar sobre el icono de "Generate SAMRAI Code", de nuevo situado en la parte superior. Luego seleccionamos el código para descargarlo, con lo que obtenemos un archivo $*.zip$ que contiene un archivo *problem.input*, con el que podremos cambiar las variables del problema.



```
Main {
        // Screen ouput information interval (In coarser level cycle units). It includes the information of all levels.
        output_interval = 1                 // zero to turn off
        // Screen information interval of execution time (In coarser level cycle units)
        timer_output_interval = 1           // zero to turn off
        // Restart configuration
        // The following parameter set if the simulation starts from a checkpoint or from scratch
        start_from_restart = FALSE
        // Interval to save checkpoints for restarting (In coarser level cycle units)
        restart_interval = 0
        // Iteration to restart from (if start_from_restart = TRUE) (In coarser level cycle units)
        restart_iteration = 0
        // Directory to store the checkpoints and to take the checkpoint for a restart
        restart_dirname = "checkpoint.restart"
        // To force a processor rebalance
        rebalance_processors = FALSE
        // dt. It should be calculated as < CFL / Vmax * (x_hi - x_lo) / Nx.
        // Being CFL a factor than depends on the time integration method. In 3D, for Runge-Kutta 3 is 0.25, for Runge-Kutta 4 is 0.4.
        // Vmax is the maximum characteristic speed.
        // Nx is the number of cells in an axis.
        dt = 0.1
        // Clustering type (See details below)
        clustering_type = "BergerRigoutsos" // TileClustering or BergerRigoutsos
        // Load Balancer (See details below)
        partitioner_type = "CascadePartitioner" // CascadePartitioner, TreeLoadBalancer or ChopAndPackLoadBalancer
}
```

Figura 13: Código generado

Por ejemplo, en la Figura 13.3 podemos ver una sección del código que incluye *dt*, es decir, el intervalo de salto temporal. Para este caso, está configurado para 0.1, pero tal vez nos podría interesar otro número por el cumnplimiento de alguna condición (de hecho tenemos que manejarlo si queremos luego cumplir la condición CFL en los esquemas que vamos a trabajar). En este caso, lo que debemos hacer es, en el archivo *problem.input*, cambiar la variable *dt* a 0.1 (en el ejemplo que nos ocupa), guardar el archivo y, a través de docker, insertarlo en el container que corresponda para así poder simular el código (esto solo en Windows).

5.5. Visualización del código

Después de todo este procedimiento ya podemos, por fin, visualizar la simulación que hemos creado a través de VisIT Studio:

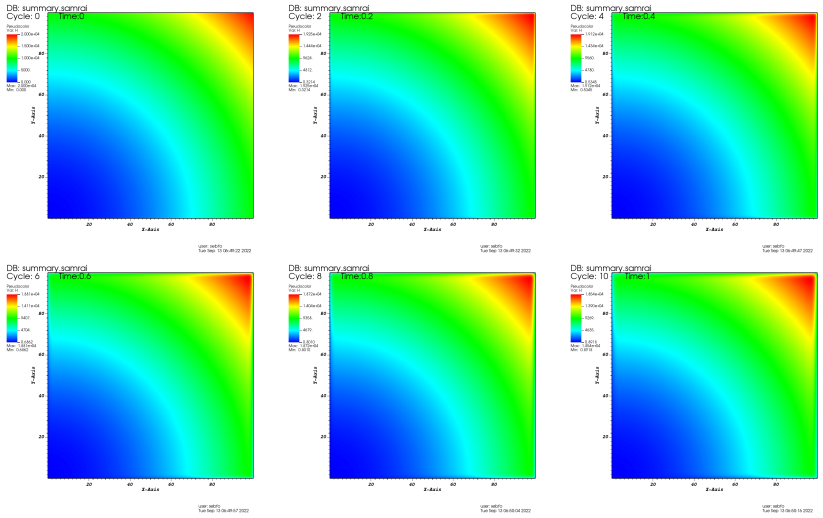


Figura 14: Visualización de la Ecuación del calor para las condiciones expuestas hasta ahora.

Queda así completado el proceso de simulación y visualización de un problema PDE.

6. Introduciendo nuevos esquemas de discretización

6.1. Ecuación de la adveccción en 1-D

Se va a proceder, para empezar, con una demostración bastante sencilla, que consiste en la visualización de la solución a la Ecuación de advección:

$$u_t + a * u_x = 0$$

Siendo $a = 1$ y, además, estableciendo condiciones de contorno periódicas y las siguientes condiciones iniciales:

$$u(x,0) = 2e^{\frac{(x-50)^2}{2*20^2}} - 2e^{\frac{(x-30)^2}{2*10^2}} + e^{\frac{(x-58)^2}{2*4^2}}$$

Dando lugar así a las siguientes condiciones iniciales:

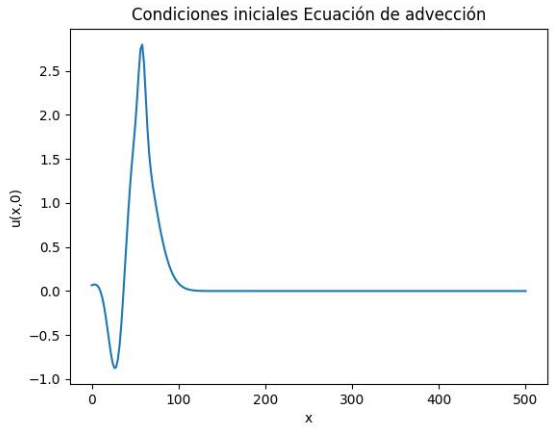


Figura 15: Condiciones iniciales de la ecuación de advección.

Siguiendo los pasos indicados en la anterior Sección, hemos definido el Modelo de la Ecuación de advección, además de las debidamente referenciadas condiciones iniciales. El esquema de discretización a usar será el siguiente, un sencillo esquema de diferencias finitas centradas de segundo orden:

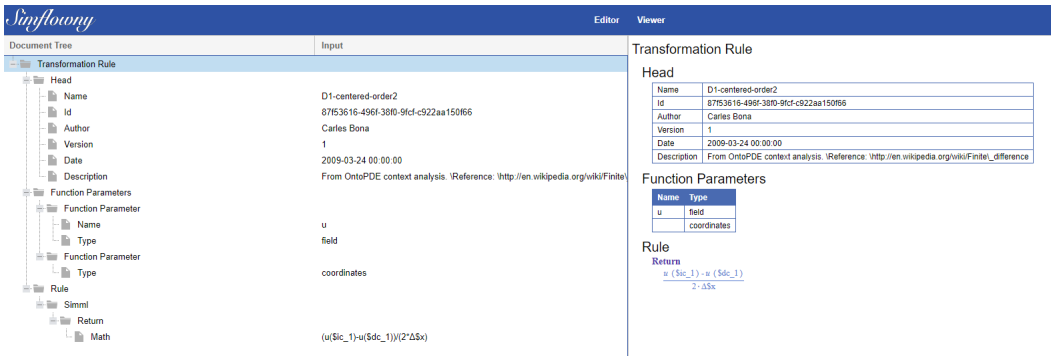


Figura 16: Imagen del esquema en la plataforma.

Vamos a comprovar la convergencia del método, a la vez que comparamos la evolución del desplazamiento de la onda para distintos valores de Δx y Δt. Para hacerlo, vamos a comparar numéricamente la evolución de la onda para tres valores distintos tanto de Δx como de Δt. Vamos a doblar el tamaño de Δx y reducirlo a la mitad, con lo que con tal de mantener la condición CFL, según la cual $\Delta t \leq \Delta x$, entonces vamos también a reducir Δt a la mitad y doblar el valor de Δt, respectivamente para cada valor de Δx. Con ello, llegamos a las siguientes figuras:

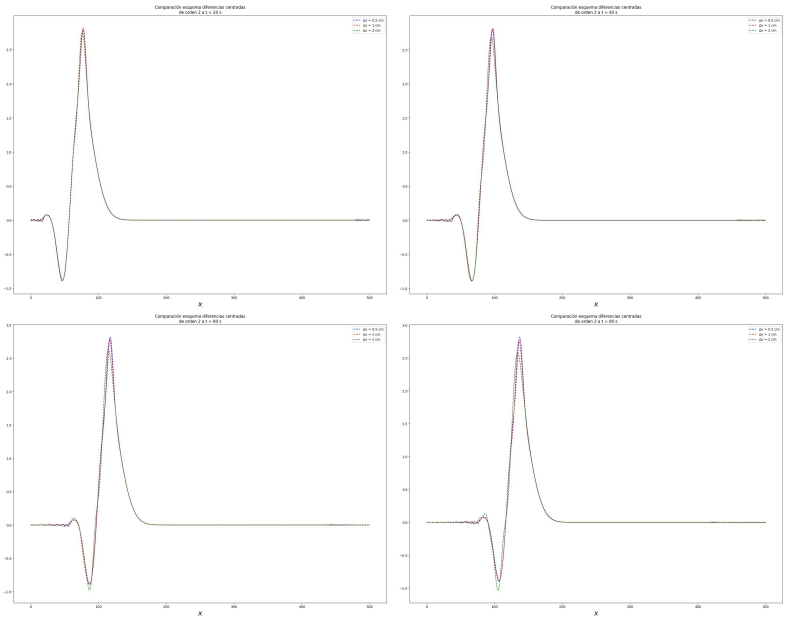


Figura 17: Comparación de la evolución de la onda para distintos valores de Δt y Δx

Se observa que, para menor Δt y Δx el error de amplitud disminuye de forma bastante cualitativa. Este hecho se puede observar en el siguiente gráfico, donde comparamos la evolución temporal del error según los distintos valores usados:

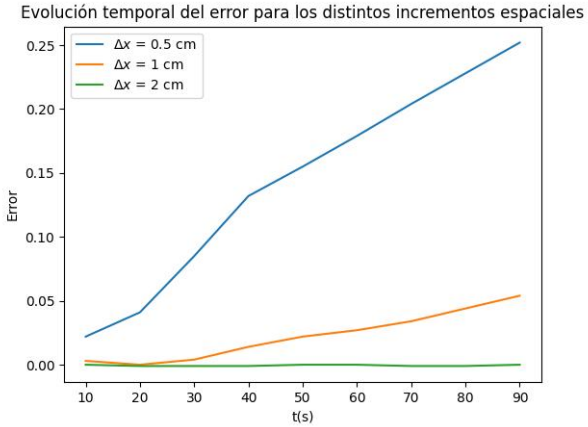


Figura 18: Comparativa del error para los distintos casos.

Esta Figura demuestra que, efectivamente, hay un factor de 4 entre la amplitud inicial y la amplitud final observadas para la discretización espacial de mayor valor. Esto corrobora que, efectivamente, se trata de un esquema de orden 2 en lo que respecta al error de amplitud.

6.2. Ecuación del calor

En la Sección donde introducíamos la plataforma, así como todas sus funciones y la preparación de los esquemas numéricos, hemos puesto como ejemplo la Ecuación del calor. Sin embargo, el ejemplo era extremadamente sencillo y se trataba sencillamente de 1 segundo, lo cual es insuficiente para realizar cualquier tipo de análisis. Por ello, ahora vamos a solucionar la Ecuación bajo las mismas condiciones, pero aumentando el parámetro *tend* a 500 s. El resultado obtenido se observa en la siguiente imagen:

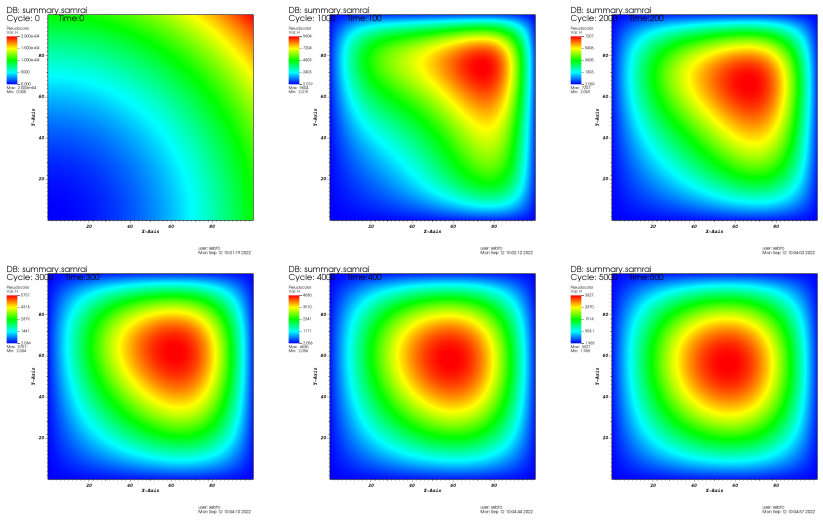


Figura 19: Evolución de la Ecuación del Calor durante 500 s.

Si cambiamos los intervalos de discretización temporal y espacial, llegamos a la siguiente figura:

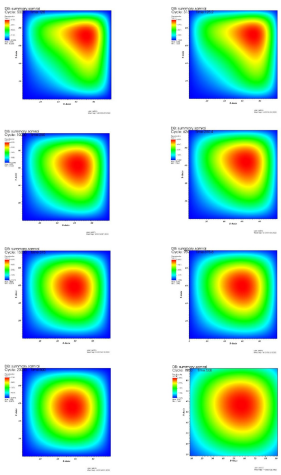


Figura 20: Evolución de $u(x, y)$ para distintos pasos de tiempo y espacio. A la izquierda, $\Delta x = 0{,}5$ y $\Delta t = 0{,}025$; a la derecha, $\Delta x = 2$ y $\Delta t = 0{,}4$

Es importante notar que, al haber escogido un tiempo relativamente bajo, los efectos de la discretización de menor resolución (la columna derecha) se notan mucho, sobre todo en las etapas finales. La columna izquierda, por el contrario, muestra un contorno mucho más definido que el de la Figura 19, con lo que queda corroborada la convergencia del esquema a un valor límite.

6.3. Ecuaciones elípticas

La plataforma no permite la resolución directa de ecuaciones elípticas o de diagnóstico. Sin embargo, este escollo puede ser superado de forma bastante sencilla. Si escogemos una ecuación (por ejemplo la del calor, presentada previamente) podemos realizamos el suficiente número de iteraciones temporales como para que se llegue a una solución estacionaria. Por ejemplo, vamos a intentar resolver el caso más sencillo de este estilo, la Ecuación de Laplace.

Por ejemplo, si en la Subsección anterior resolvíamos la Ecuación del calor, en este caso vamos a resolver cual es el estado de una placa con una fuente de calor que tiene la forma $f(x) = x^2 + x$ en el extremo vertical inferior, mientras que en el resto de bordes imponemos condiciones de contorno nulas.

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0$$

Por lo tanto, las condiciones de contorno serían las siguientes:

1. $u = x^2 + x$ en $y = 0$ cm.

2. $u = 0$ en $y = 300\ cm$.

3. $u = 0$ si $x = 0\ cm$.

4. $u = 0$ si $x = 300\ cm$.

Siguiendo los pasos explicados en la Sección 5, diseñamos el problema en la plataforma, estableciendo las condiciones de contorno que correspondan. Llegamos a los siguientes resultados:

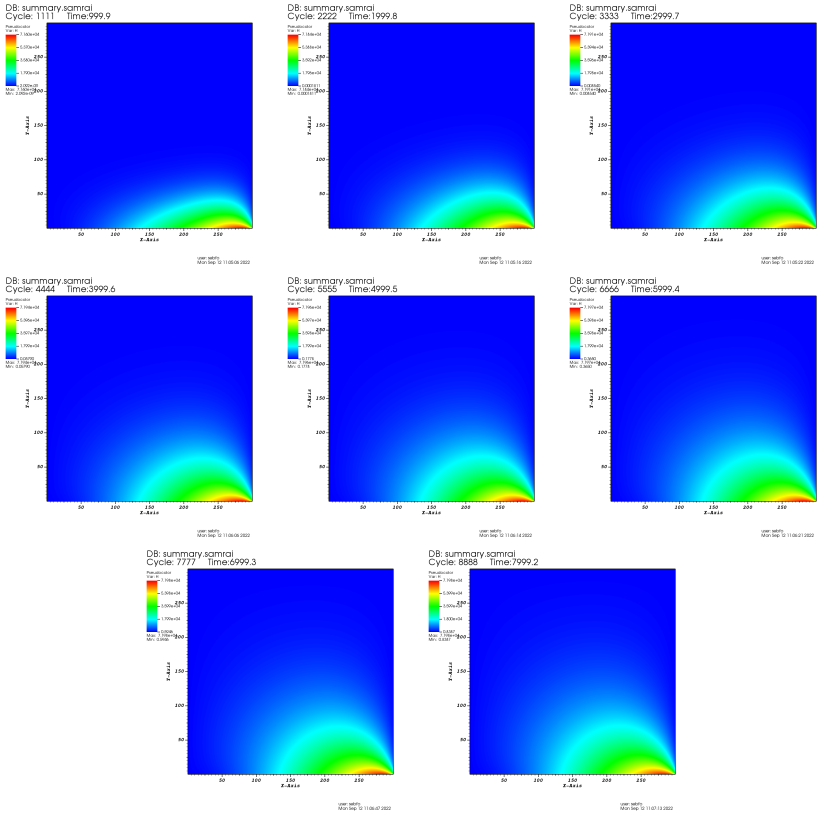


Figura 21: Evolución de la Ecuación a través del tiempo hasta alcanzar un estado estacionario.

Es obvio que hemos establecido un tiempo límite de 8000 segundos. Para cerciorarnos de que realmente se ha alcanzado un estado estacionario, se comparará a través de la siguiente figura (que es una representación gráfica de una de las secciones horizontales del gráfico) que, para $t \to 8000\ s$, la solución efectivamente se ha estabilizado:

También comprobaremos la validez del esquema, de nuevo, comprobando distintos intervalos que den lugar a soluciones con mayor o menor resolución.

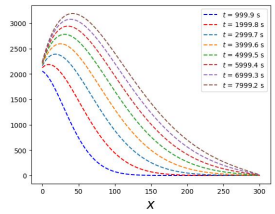


Figura 22: Comparación para distintos tiempos de la forma de la sección horizontal $u(x, y = 45)$.

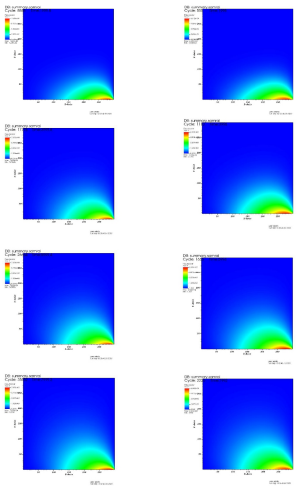


Figura 23: Evolución de $u(x, y)$ para distintos pasos de tiempo y espacio. A la izquierda, $\Delta x = 1{,}5$ y $\Delta t = 0{,}225$; a la derecha, $\Delta x = 6$ y $\Delta t = 3{,}6$

De nuevo, queda patente como observación general que los valores de pequeños de Δt y Δx dan lugar a soluciones más próximas a un valor límite. También es necesario comentar que, al igual que cuando hemos analizado la Ecuación del calor, los intervalos de discetrizaciones mayores dan lugar a formas más difusas, si bien no parecen tener ningún efecto notable a la hora de analizar el tiempo que tardan en dar lugar a soluciones estacionarias, algo lógico considerando la magnitud de los intervalos temporales respecto al orden temporal en que estamos trabajando (de hasta 8000 segundos).

7. Mi experiencia con la plataforma.

Mis primeros pasos con Simflowny fueron bastante sencillos, dentro de lo que cabe, gracias a que los tutoriales y demás herramientas que me daban los desarrolladores eran claros y concisos. A pesar de que el uso de Docker era una experiencia nueva para mí, no hace falta un conocimiento muy profundo de la aplicación para usar la plataforma. Más tarde, durante la primera toma de contacto con la plataforma, también quedé impresionado por la simplicidad de edición que ofrecía. A pesar de proponer la creación de problemas y modelos más complejos de los que tal vez yo podría llegar a usar, uno podía trabajar con Simflowny de forma intuitiva... hasta cierto punto, claro está, puesto que también hay que decir que de alguna forma tantas opciones al principio más temprano de mi toma de contacto fueron un poco abrumadoras.

Para ayudar con esta experiencia, estaba a mi disposición un tutorial: "Dam break with bed". Este documento me permitió entender la visualización del código, puesto que sencillamente consistía en una discretización ya preparada (como la que vimos en la Subsección 5.3) que se tenía que llevar a cabo y luego ejecutar el código que ya estaba preparado. En ese momento me encontré con mi primer obstáculo, puesto que en Física Computacional (así como en Física Asistida por Ordenador, que es donde tomamos contacto por vez primera con el lenguaje de programación Python) no usamos en ningún momento una herramienta similar a lo que requería la plataforma, que eran unos conocimientos de Ubuntu que yo desconocía, por lo que tuve que pedir ayuda. Gracias a Docker, pude simular el entorno de Ubuntu en Windows y pude finalizar el procedimiento satisfactoriamente, tal y como indica la figura:

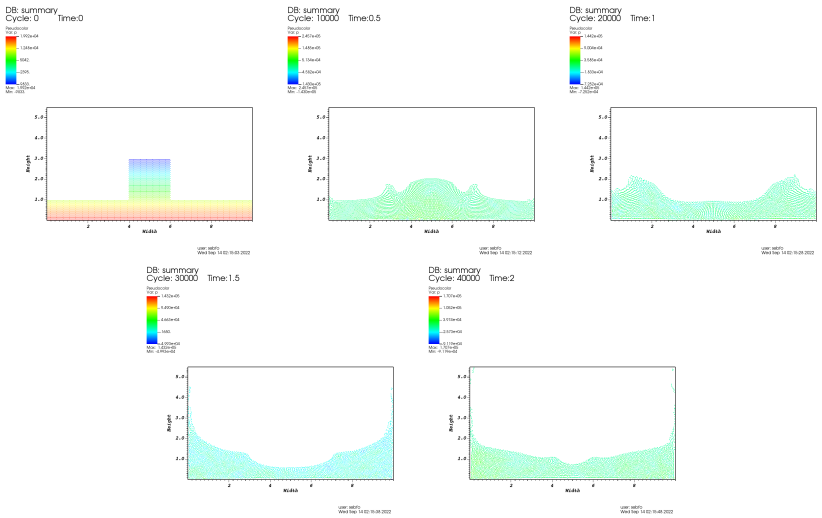


Figura 24: Gráfico obtenido para la discretización "Dam break with bedz su consiguiente evolución en el tiempo.

Seguidamente subí un nivel a la hora de preparar el problema. Por primera vez no me limité a ejecutar una discretización ya preparada, sinó que tenía que diseñar un problema en el que tenía que introducir las condiciones inciales y de contorno necesarias, tal y como he indicado en la segunda etapa del proceso de resolución del problema. Teniendo en cuenta que en esta etapa ya estaba superado el problema de la simulación por falta de familiaridad, no hubo muchos problemas y pude llevar a cabo el problema prediseñado, que consistía en una onda que se expandía bidimensionalmente, estando limitada por unas condiciones de contorno periódicas. Obtuve el siguiente resultado:

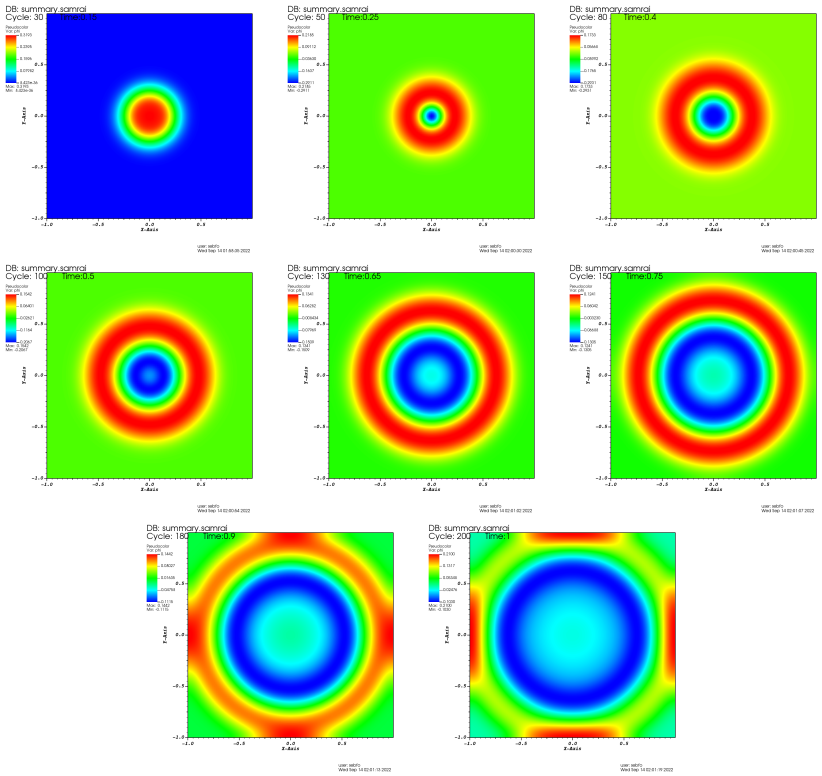


Figura 25: Ecuación de la onda en 2D, para diferentes instantes de tiempo.

Después de este último tutorial, también amablemente cedido por los desarrolladores, ya estaba preparado para empezar el diseño de un problema desde su etapa más temprana. A pesar de no haber muchos problemas en este sentido, cabe criticar que la plataforma no notificaba la falta de algunos parámetros que eran imprescindibles (como el modelo interior a usar), dando lugar a errores extraños y sin explicación que me atascaban, puesto que no me informaban correctamente de dónde estaba el origen del error.

Por suerte, al hablarlo con los creadores se dieron cuenta de ello y fue so-

lucionado rápidamente, con lo que ya podía dar inicio a la visualización de un problema creado desde 0 (y para comprobar los resultados obtenidos basta comprobar lo presentado en la Sección 5, donde explico el procedimiento de creación y visualización de un problema).

Finalmente, podía dar inicio a la implementación de esquemas, siendo mi intención usar los que vimos en Física Computacional. Lamentablemente, esta etapa no llegó a un buen término. Para Upwind y Lax-Wendroff, por ejemplo, llegamos a las siguientes soluciones, siendo las condiciones iniciales las mismas que en la Figura 15:

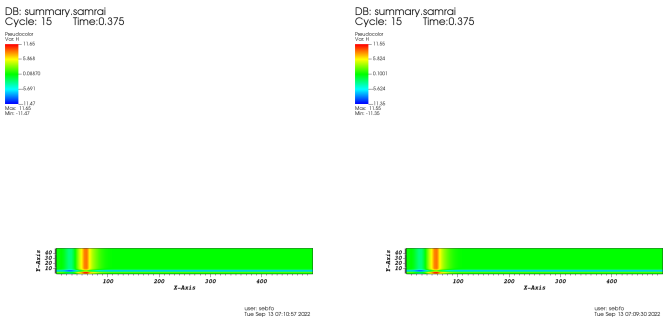


Figura 26: Comparación de los resultados obtenidos para Upwind(izquierda) y Lax-Wendroff(derecha).

Las discontinuidades observadas eran muy extrañas, no solo porque no se desplazaban, sinó también porque aparecían asimetrías en el eje y que no tienen motivo alguno para aparecer, puesto que la discretización tendría que actuar solamente a lo largo del eje x, que era a través del cual habíamos definido la derivada. Esto era uno de los aspectos que más me interesaban del proyecto, con lo que fue una pena no poder localizar el origen de este error, y al final se tuvo que proceder a un simple esquema centrado en 2 puntos, el que fue usado para obtener los gráficos presentados en la Subseción. En lo que respecta a los 2 otros porblemas, no hubo incidencias ningún tipo, puesto que me encontraba en un punto en el que me sentía muy familiarizado tanto con el entorno de Ubuntu com con la plataforma, con lo que a nivel de planteamiento no surgieron errores, y en lo que respecta a la visualización, aunque sí que hubo que realizar cambios menores como cambios en el intervalo temporal y espacial para respetar la condición CFL, no merecen apenas mención.

8. Conclusiones

Las soluciones obtenidas, así como mi experiencia con el uso de la plataforma, nos permiten llegar a las siguientes conclusiones:

Primero, es importante destacar que las discretizaciones de la plataforma han dado lugar a soluciones satisfactorias: las Figuras nos demuestran que, efectiva-

mente, cuanto menor es el incremento espacial o temporal usado, las soluciones convergen a un valor límite, el de la solución real. La ecuación del calor también evoluciona satisfactoriamente, y en el caso de la Ecuación de Laplace hemos llegado a una solución estacionaria, tal y como se pretendía.

Sin embargo, lamentablemente no se ha podido cumplir el objetivo deseado respecto a los esquemas de Física Computacional. El método Upwind y Lax-Wendroff no han funcionado debido a problemas con la plataforma, pero el método Leap-Frog se ha tenido que descartar debido a que, por concepto, no es aplicable al método de líneas, puesto que requiere del uso de pasos intermedios en la malla temporal. La discretización usada para la Ecuación del calor puede ser interpretada como un caso concreto del método Θ, pero no se ha conseguido generalizar el caso. En definitiva, los resultados son los deseados pero no de la forma que se deseaba.

El funcionamiento de la plataforma, a pesar de que al principio fue algo tosco debido a mi falta de contacto con *Docker* y el entorno *Ubuntu* ha sido en líneas generales muy satisfactorio. Una vez familiarizado, no costaba nada cambiar las condiciones que deseara dentro de $Simflowny$, o editar el código directamente desde una ventana de comandos en Windows. Aun así, tengo que decir que esta creciente familiaridad se ha debido sobre todo al enorme apoyo que han supuesto tanto mi tutor como los creadores de la plataforma, con lo que en un contexto diferente (el de un estudiante que no tenga mucha experiencia con trabajar directamente con el sistema operativo, como yo mismo) tal vez habría resultado algo confuso. Incluso contando con ello, este tipo de problemas se solucionan perfectamente cambiando elementos de los tutoriales de los que ya disponen que permitan una mayor accesibilidad a perfiles más amplios. Puntualmente también han surgido problemas como la falta de $feedback$ por parte de la plataforma en sí a la hora de informar de ciertos problemas, pero los creadores han sido informados y no volvieron a ocurrir.

En conclusión, $Simflowny$ tiene muchísimo por ofrecer, puesto que presenta una GUI intuitiva y muy sencilla de usar, y cuyos problemas se derivan más de una posible falta de información a personas que no estén acostumbradas a trabajar con *Ubuntu*, hecho del que sería injusto responsabilizar del todo a los creadores. Las soluciones obtenidas son satisfactorias y permiten una personalización rápida y eficaz de las condiciones del problema, sean estas referentes a su planteamiento o a su aspecto más ligado a la computación, como el incremento espacial o temporal usados. Es, sin duda alguna, una herramienta muy útil a la hora de trabajar tanto con EDPs como con esquemas numéricos.

9. Bibliografía

- https://bitbucket.org/iac3/simflowny/wiki/UserGuide
- Apuntes de Física Computacional durante el curso 2020-2021, a cargo de Víctor Homar.
- Schiesser, W. E. (1991). The Numerical Method of Lines.