Simulación numérica directa de turbulencia homogénea. Paralelización de un método pseudoespectral.

Ignacio López Díaz

A mis padres por su cariño y apoyo,
a Patricia por su amor y paciencia,
y a todos aquellos que me hacen
sentir la vida.

Quisiera expresar mi agradecimiento a todos los que me han ayudado a la elaboración del proyecto. Especialmente al Dr. Javier Dávila por su tutela y colaboración y al Dr. Manuel Contreras por su ayuda en la correcta formulación matemática, así como a todo el Área de Mecánica de Fluidos de la Escuela Técnica Superior de Ingenieros de Sevilla por su apoyo durante el desarrollo del proyecto.

Índice general

Capítulo 1

Resumen del proyecto y nomenclatura

El presente proyecto fin de carrera se encuadra dentro del proyecto de investigación del Plan Nacional titulado "Análisis del flujo, sedimentación de partículas y oxigenación en tanques de producción de peces y crustáceos. Optimización y parámetros de diseño" del Grupo de Mecánica de Fluidos de la Universidad de Sevilla.

1.1. Descripción del problema

El objetivo de este proyecto es realizar una simulación de la evolución temporal de un flujo turbulento homogéneo e isótropo, en un cubo extendido periódicamente en las tres coordenadas espaciales, que por comodidad se ha elegido de lado 2π. Para esta simulación se va a emplear un cluster (ver capítulo 5) que nos permite superar las limitaciones intrínsecas a esta simulación. Dado que el campo de estudio es un cubo, para poder simularlo dividimos cada lado en N puntos en los que vamos a realizar los cálculos. Por tanto, tenemos necesidades de memoria del orden de N^3, lo que significa que las simulaciones requerirán de grandes recursos de memoria y los cálculos serán muy costosos en tiempo ya que con un N típico de valor 128 con precisión simple las necesidades son del orden de 8MB por matriz y en los cálculos se usan por encima de 12 matrices.

En la simulación del flujo turbulento se va a usar un método pseudoespectral [22] en el que se resuelven las ecuaciones de Navier-Stokes utilizando el espacio físico y el de Fourier de forma que se minimicen los costes computacionales.

Para la paralelización del código se va a utilizar el estándar MPI (Message Passing Interface). Su uso ofrece una interfaz que gestiona los procesos de sincronización e intercambio de información entre los distintos nodos del cluster.

1.2. Estructura del proyecto

Este proyecto se va a estructurar en diferentes capítulos que cubren los distintos aspectos de su elaboración. En el capítulo 2 se hace una breve introducción a la turbulencia en el que se describe las propiedades de la misma así como una descripción de los conceptos y parámetros que vamos a utilizar. En el capítulo 3 se hace una breve introducción al análisis de Fourier ya que es la base sobre la que se apoya el desarrollo teórico de este proyecto. En él se describe su expresión matemática así como las propiedades que se van a utilizar. En el capítulo 4 se hace el desarrollo teórico del método pseudoespectral que se utiliza para obtener las ecuaciones

que nos proporcionan la evolución temporal del campo de velocidades en el cubo. En el capítulo 5 se hace una descripción del cluster que se usa para ejecutar el código. Se indican tanto sus características como los pasos necesarios para su correcta configuración. En el capítulo 6 se analiza la resolución del problema. Se describen los conceptos más importantes de la simulación, se da una explicación de la implementación del código y del sistema de configuración en tiempo real y, por último, se explican los puntos a los que se ha aplicado la paralelización. En el capítulo 7 se explican los resultados que ofrece el programa y que proporcionan una descripción del estado instantáneo del campo de velocidades, así como una serie de estadísticas que lo caracterizan, como son el espectro, la gráfica de invariantes u otros parámetros. Finalmente, en el capítulo 8 se hace una breve descripción del proyecto y las posibles líneas de investigación que podrían ampliarlo.

1.3. Nomenclatura

En este documento, usamos la siguiente nomenclatura:

- Escalares: Texto normal en cursiva (por ejemplo, ν).
- Números adimensionales: Texto normal (por ejemplo, Re).
- Vectores: Texto en negrita (por ejemplo, $\mathbf{v}$).
- Componente de un vector: Texto con separación y subíndice ($\mathbf{v}|_j$).
- Serie de Fourier: Texto con llave superior (por ejemplo $\widehat{\mathbf{f}}$ es la serie de Fourier de la función vectorial $\mathbf{f}$).
- Coeficientes de una serie de Fourier: Texto con llave superior y subíndice (por ejemplo $\widehat{\mathbf{f}}_k$ es el k-ésimo coeficiente de la serie de Fourier de $\mathbf{f}$).
- Valor medio en el espacio: Texto entre los símbolos $\langle\cdot\rangle$. Por ejemplo, el valor medio de f en el espacio se calcula:

$$\langle f\rangle = \frac{1}{N^3}\sum_{n_1=0}^{N-1}\sum_{n_2=0}^{N-1}\sum_{n_3=0}^{N-1} f(n_1,n_2,n_3). \tag{1.1}$$

Capítulo 2

Notas de turbulencia

2.1. Introducción

El estudio de la turbulencia es un campo de gran interés y aplicación a un gran número de disciplinas científicas y tecnológicas. Entre otras se pueden citar los estudios metereológicos, los de los movimientos de los mares y océanos, los de los chorros que salen de un grifo con suficiente velocidad, los de los gases en el interior de un motor, e incluso los de la respiración en los seres humanos.

No se ha encontrado todavía un modelo que permita obtener información precisa de la evolución de un fluido turbulento mediante una resolución matemática sencilla, por eso, se recurre a la simulación numérica para su estudio. El principal problema de estas simulaciones es el gran número de recursos computacionales necesarios para los casos de interés práctico. Por este motivo la tendencia actual es realizar simulaciones numéricas con las escalas grandes y usar modelos que representen fielmente las escalas pequeñas de forma que se reduzca el número de cálculos numéricos. Es en este campo donde todavía hay que trabajar bastante ya que los modelos actuales no ofrecen resultados satisfactorios sobre todo en problemas de dispersión y sedimentación de partículas.

Dado que, como veremos, en las escalas pequeñas se puede utilizar una aproximación de turbulencia homogénea e isótropa, este proyecto se enmarca en la línea de buscar los modelos que establezcan la dinámica de las escalas pequeñas para poder integrarlos en simulaciones de interés real.

2.2. Turbulencia

En un flujo de un fluido el número de Reynolds (Re) establece una relación entre la fuerza inercial y la fuerza de la viscosidad. Por debajo de un determinado número de Reynolds se habla de flujos laminares que tienen soluciones analíticas y numéricas sencillas modelando con una formulación matemática como la de las ecuaciones de Navier-Stokes [1]. A partir de un número de Reynolds crítico el flujo se vuelve inestable y si sigue creciendo da lugar a un flujo turbulento. En la figura 2.1 se observa como un flujo laminar debido a una inestabilidad se hace inestable produciéndose la transición a un flujo turbulento.

Resulta complicado, a pesar de que se tiene una idea intuitiva, definir la turbulencia. Vamos a expresar en cambio alguna de sus propiedades.

- Tiene una distribución espacial de las variables fluidas muy irregular. Además, su evolución temporal también lo es.

Figura 2.1: Flujo turbulento.

- Es esencialmente tridimensional.
- Es muy disipativa necesitando de un aporte de energía exterior para mantenerse en el tiempo.
- Es extremadamente compleja al coexistir en ella movimientos del fluido con una disparidad enorme de longitudes características (también de tiempos), si se entiende por tales las distancias que hay que recorrer en el fluido para que se produzcan variaciones apreciables de las magnitudes fluidas.
- Es un proceso caótico, de forma que para poder predecir la evolución de los sistemas sería necesario una precisión en las condiciones iniciales imposible de obtener experimental o numéricamente.
- Esta gran sensibilidad a las condiciones iniciales y su irregularidad hacen que estos flujos sean casi aleatorios y que su simulación directa con las ecuaciones de Navier-Stokes sea extremadamente costosa.

A consecuencia de estas características, para analizar los flujos turbulentos se recurre generalmente a la estadística. Dada una variable fluida, si se promedia en un intervalo de tiempo suficiente, los valores medios locales obtenidos se comportan de forma determinista, variando en el espacio y tiempo mucho más suavemente que los valores instantáneos. Por tanto, el objetivo principal de los métodos analíticos y numéricos empleados en turbulencia es el cálculo de las magnitudes medias del flujo. De este planteamiento surge el problema de cierre de la turbulencia que, a pesar del esfuerzo realizado en más de medio siglo, no se ha resuelto aún.

2.3. Cascada de Kolmogorov

A partir de los trabajos de Kolmogorov[†] aparece el modelo de *cascada de energía*, en el que se postula que la turbulencia está formada por torbellinos de diferentes tamaños. Los torbellinos más grandes, que tienen una longitud característica del orden del dominio

[†]Andrei Nikolaevich Kolmogorov (1903-1987) completó sus estudios en la Universidad Estatal de Moscú en 1925 donde llegó a ser profesor en 1930. En 1935 recibió el doctorado en física y matemáticas; y desde 1938 hasta su muerte mantuvo la cátedra en el Departamento de Lógica Matemática. Fue miembro de la Academia de Ciencias y Ciencias Pedagógicas de la Unión Soviética y de la Academia de Ciencias de Estados Unidos además de recibir varios premios internacionales.

fluido, se vuelven inestables y van transfiriendo energía a torbellinos más pequeños que a su vez también la van transmitiendo a torbellinos aún más pequeños creando la llamada *cascada de energía*. Esta cascada de energía va creando torbellinos pequeños a partir de los más grandes hasta llegar a una escala en la que los efectos de la viscosidad la disipan. Se puede comprobar que en las escalas grandes, el tiempo característico de variación del movimiento de los torbellinos, puede estimarse a partir de su longitud característica (L) y la velocidad característica (u') de las fluctuaciones turbulentas de velocidad, determinadas por las condiciones de contorno impuestas al sistema (caudal, diferencias de presiones, etc.). Las escalas grandes tienen una gran dependencia de las condiciones iniciales y de contorno, pero como en un flujo turbulento el número de Reynolds es muy alto, los valores característicos asociados a los torbellinos pequeños son mucho menores que los de los grandes, por este motivo el movimiento asociado a las escalas pequeñas resulta estadísticamente independiente de las condiciones de iniciales y de contorno. La principal aportación del modelo de Kolmogorov es que predice adecuadamente la distribución de energía entre las diferentes escalas, esto es, en estado estacionario la transferencia de energía entre todos los tamaños de torbellinos tiene que ser la misma e igual a la que se inyecta a través de los mayores.

Si definimos u_l como la velocidad característica de un torbellino de tamaño l tenemos que en un sistema en equilibrio la transferencia de energía se puede definir como energía ($\sim u_l^2$) dividida por el tiempo ($\sim \frac{l}{u_l}$)

$$\epsilon \sim \frac{u_l^2}{\frac{l}{u_l}} = \frac{u_l^3}{l} = \frac{u'^3}{L}. \tag{2.1}$$

Se puede usar esta definición para estimar las dos longitudes que delimitan la cascada. Normalmente se caracteriza la velocidad de los torbellinos grandes por la raíz cuadrada del valor medio de la fluctuación de la velocidad al cuadrado

$$u' = \left\langle |\mathbf{u} - \mathbf{U}|^2 \right\rangle^{\frac{1}{2}}, \tag{2.2}$$

donde $\mathbf{U}$ es el valor medio de la velocidad ($\mathbf{U} = \langle \mathbf{u} \rangle$). Esto se debe a que según 2.1 los torbellinos más grandes tienen las diferencias de velocidades más acusadas, para ciertos torbellinos se cumple exactamente la ecuación:

$$\epsilon = \frac{u'^3}{L}. \tag{2.3}$$

Esta ecuación puede usarse para definir la *longitud integral* que notamos L y que es la longitud a la que se inyecta energía al sistema.

Se pueden calcular las escalas características de los torbellinos más pequeños para los cuales la cascada puede considerarse como no viscosa. Para cada tamaño de torbellino definimos $T_\nu = l^2/\nu$ como el tiempo en el cual la viscosidad por sí sola lo disiparía. Si el tiempo de inestabilidad no viscosa de un torbellino o tiempo de residencia ($T_r = l/u_l$) es menor que T_ν, las inestabilidades lo rompen antes de que la viscosidad tenga tiempo de actuar, si el tiempo es mayor pasa lo contrario. En el caso en el que ambos tiempos sean del mismo orden, a la longitud característica de esos torbellinos la llamamos longitud de disipación de Kolmogorov (η) cuya expresión se puede obtener de igualar las expresiones de ambos tiempos:

$$\frac{\eta^2}{\nu} = \frac{\eta}{u_\eta}, \tag{2.4}$$

de esta expresión se puede despejar u_η

$$u_\eta = \frac{\nu}{\eta}. \tag{2.5}$$

Operando con la ecuación 2.3 se puede llegar a una expresión para η que sea independiente de u_η:

$$\epsilon = \frac{u_\eta^3}{\eta} = \frac{\left(\frac{\nu}{\eta}\right)^3}{\eta} = \frac{\nu^3}{\eta^4}. \tag{2.6}$$

Reordenando llegamos a

$$\eta = \left(\frac{\nu^3}{\epsilon}\right)^{\frac{1}{4}}. \tag{2.7}$$

A partir de la longitud de disipación de Kolmogorov podemos dividir los torbellinos en dos grupos. Para los torbellinos que tengan una longitud $l \gg \eta$ la inestabilidad actúa más rápido que la viscosidad cuyos efectos son despreciables y para los casos en los que $l \ll \eta$ es la inestabilidad la que es despreciable y toda la disipación de energía se produce por efecto de la viscosidad.

El caso $l \gg \eta$ se conoce como *rango inercial* aunque este término es a veces usado para los torbellinos autosemejantes en los que también se cumple que $l \ll L$. Por otra parte, el caso $l \ll \eta$ se conoce como *rango disipativo*, en éste el flujo es suave y la velocidad puede ser aproximada por series de Taylor con el resultado de que las diferencias de velocidad son proporcionales a l. Si definimos ω' como el valor medio del rotacional de velocidades

$$\omega' = \langle \nabla \times \mathbf{u} \rangle, \tag{2.8}$$

tenemos que

$$u_l \sim \omega' l. \tag{2.9}$$

Hay que hacer notar que 2.9 no vale para el rango inercial ya que no existe un valor finito de $u_l/l \sim l^{-2/3}$ según l va tendiendo a cero y el flujo no puede ser descrito como una función diferenciable.

En la escala de Kolmogorov ($l = \eta$) podemos obtener una ecuación que nos permite calcular ϵ a partir de las ecuaciones anteriores

$$\epsilon = \frac{u_l^3}{l} = \frac{\omega'^3 l^3}{l} = \omega'^3 l^2 = \left(\omega' l^2\right) \omega'^2, \tag{2.10}$$

usando la definición de T_ν se puede simplificar la ecuación anterior

$$T_\nu = \frac{l}{u_l} = \frac{1}{\omega'} = \frac{l^2}{\nu}, \tag{2.11}$$

lo que implica que $\omega' = \dfrac{\nu}{l^2}$ y llegamos a

$$\epsilon = \nu \omega'^2. \tag{2.12}$$

En el cuadro 2.1 resumimos las características de los distintos rangos de torbellinos para una turbulencia isotrópica.

Coste computacional

Para hacer una estimación del coste computacional de flujos turbulentos típicos en los que $Re \sim 10^6$ vamos a ver el número de cálculos necesarios.

Rango	**Longitud**	**Velocidad**	**Gradiente de v**	**Tiempo de disipación**
Torbellinos grandes	L	u'	$u'/L = \epsilon/u'^2$	$T_I = L/u' = u'^2/\epsilon$
Inercial	l	$u_l \approx 1{,}4(\epsilon l)^{1/3}$	$u_l/l \sim (\epsilon/l^2)^{1/3}$	$(l^2/\epsilon)^{1/3}$
Kolmogorov	$\eta = (\nu^3/\epsilon)^{1/4}$	$u_\eta = (\epsilon\nu)^{1/4}$	$\omega_\eta = (\epsilon\nu)^{1/2}$	$T_\eta = \omega'^{-1} = (\nu/\epsilon)^{1/2}$
Disipativo	l	$\omega' l$	ω'	$T_\nu = l^2/\nu$

Cuadro 2.1: Resumen de las características de los rangos.

En una simulación determinada sería necesario realizar una división del espacio en una malla tridimensional de N^3 elementos, donde $N \sim \frac{L}{\eta}$. Para relacionar el valor de N con el número de Reynolds sustituimos 2.3 en 2.7 y operamos para obtener

$$\eta = \left(\frac{\nu^3}{\frac{u'^3}{L}}\right)^{1/4} = \left(\frac{\nu^3 L^3 L}{u'^3 L^3}\right)^{1/4} = \left(\frac{\nu}{u'L}\right)^{3/4} L. \tag{2.13}$$

Como $Re = \frac{u'L}{\nu}$ llegamos a la relación

$$\frac{L}{\eta} = Re^{3/4}. \tag{2.14}$$

A partir de esta expresión podemos determinar el orden de magnitud del número de puntos en el mayado del cubo

$$N^3 \sim \left(\frac{L}{\eta}\right)^3 = Re^{9/4}. \tag{2.15}$$

A continuación vamos a ver las necesidades de cómputo en una simulación de un tiempo integral. El tiempo integral lo tomamos como el tiempo característico de disipación de las escalas grandes (T_I), establecemos que los pasos de integración (iteraciones temporales) no pueden ser superiores al menor de los tiempos característicos del problema, es decir, T_η. Por tanto, el número de pasos integrales es

$$N_T = \frac{T_I}{T_\eta} \sim \frac{L/u'}{(\nu/\epsilon)^{1/2}} = \frac{L/u'}{\left(\frac{\nu}{u'^3/L}\right)^{1/2}} = \frac{u'^{\frac{1}{2}} L^{\frac{1}{2}}}{\nu^{\frac{1}{2}}} = Re^{1/2}. \tag{2.16}$$

Por otra parte se ha estimado que el número de operaciones del procesador en un paso de integración por punto en simulaciones de este tipo es del orden de 10^3, por lo que el número total de operaciones en un tiempo integral en todo el cubo es

$$10^3 N^3 N_T \sim 10^3 Re^{9/4} Re^{1/2} = 10^3 Re^{11/4}. \tag{2.17}$$

Esto significa que para un valor típico del número de Reynolds ($Re = 10^6$) el número de operaciones es extremadamente alto ($10^3 10^{6\frac{11}{4}} = 10^{19{,}5}$), un ordenador de 1 Gflop de velocidad tardaría más de 1000 años en realizar la simulación, incluso un superordenador de 1 Tflop tardaría del orden de un año. Este es el motivo principal de buscar modelos que simulen el comportamiento de las escalas pequeñas. Para reducir las necesidades computacionales se intenta utilizar sistemas que simulen numéricamente las escalas grandes, que tienen una influencia mucho más acusada de las condiciones iniciales y de contorno, y que calculen las escalas pequeñas mediante modelos simplificados.

Parámetros de interés

Por último, vamos a indicar las fórmulas de algunos parámetros que se han calculado para su comparación con simulaciones conocidas con el fin de comprobar la exactitud de los cálculos.

Escala de Taylor: La escala de Taylor no tiene una interpretación física clara. Se define como

$$\lambda = \sqrt{\frac{15\nu u'^2}{\epsilon}}. \tag{2.18}$$

Número de Reynolds en la escala de Taylor: Se suele usar para caracterizar la turbulencia de homogénea e isótropa.

$$Re_\lambda = \frac{u'\lambda}{\nu}. \tag{2.19}$$

Relación dt/T_η: Este valor está relacionado con el tamaño del paso que se puede tomar para que el error se mantenga dentro de ciertos límites. Puesto que se va a usar un método de segundo orden, el error es proporcional a dt^2. Como las escalas más pequeñas tienen tiempo característicos T_η, vamos a mantener el paso de integración en el intervalo $0{,}015 \leq \frac{dt}{T_\eta} \leq 0{,}025$ de forma que se mantiene un compromiso entre el error cometido y la velocidad de computación.

2.4. Espectro

El análisis de Fourier es un campo bien desarrollado y conocido de gran aplicación en la ingeniería. En esta sección explicaremos someramente su aplicación a la representación de flujos turbulentos.

El número de onda asociado con una longitud de escala l se define como $k = \frac{2\pi}{l}$. A partir de esta definición, mediante la transformada de Fourier, se transforma el campo de velocidades $\mathbf{u}$ en el espacio físico en el espacio de Fourier $\widehat{\mathbf{u}}_k$, donde está descrito en función de los números de onda. Esta transformación nos permite realizar cálculos sobre la energía del sistema de forma más cómoda, expresando la distribución de energía entre los torbellinos en términos de su espectro. El espectro es una función que resume la energía asociada a cada número de onda independientemente de su distribución espacial. Una interpretación del significado físico del espectro es que $E(k)dk$ es la energía cinética por unidad de masa contenida en las escalas con números de onda comprendidos entre k y $k + dk$. Debido a su pequeña longitud característica, el flujo en el rango inercial debe ser independiente de los detalles del flujo en la escala grande y sólo depende de éste a través de la energía disipada por unidad de masa y tiempo ϵ. Aplicando el teorema π de análisis dimensional, como $E = E(k, \epsilon, \nu)$ y para $l \gg \eta$ no depende de ν, llegamos a $\frac{E(k)}{\epsilon^{2/3}k^{-5/3}} = c_k$. Se ha comprobado que c_k es una constante universal cuyo valor aproximado es $c_k = 1{,}5$.

Para extender el espectro de Kolmogorov e incluir tanto la dependencia de las escalas más grandes ($\sim L$) como de las más pequeñas ($\sim \eta$), se ha obtenido a partir de resultados experimentales [19] una expresión que modela la forma del espectro

$$E(k) = c_k \epsilon^{\frac{2}{3}} k^{-\frac{5}{3}} f_L(kL) f_\eta(k\eta), \tag{2.20}$$

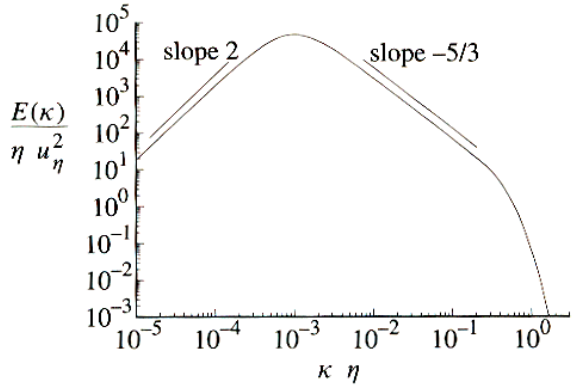

Figura 2.2: Modelo del espectro.

donde

$$f_L(kL) = \left(\frac{kL}{\left[(kL)^2 + c_L \right]^{\frac{1}{2}}} \right)^{\frac{5}{3}+2} \tag{2.21}$$

y

$$f_\eta(k\eta) = e^{-\beta\left\{ \left[(k\eta)^4 + c_\eta^4 \right]^{\frac{1}{4}} - c_\eta \right\}}. \tag{2.22}$$

Las constantes que se usan en las fórmulas anteriores (obtenidas experimentalmente) son: $\beta = 5{,}2$, $c_L \approx 6{,}78$ y $c_\eta \approx 0{,}40$. Se puede ver una representación gráfica de este modelo en la figura 2.2.

2.5. Invariantes

Para el análisis de las variaciones locales del campo de velocidades se suele recurrir al cálculo de los invariantes. El motivo es que si se aproxima el vector de velocidad por un desarrollo en serie de Taylor y nos quedamos con los primeros términos obtenemos

$$\mathbf{u}\left(\mathbf{x}, t\right) \simeq \mathbf{u}\left(\mathbf{x_0}, t\right) + \nabla \mathbf{u}|_{\mathbf{x}=\mathbf{x_0}} \left(\mathbf{x} - \mathbf{x_0}\right). \tag{2.23}$$

La parte que depende de las variaciones del campo de velocidades es $\nabla \mathbf{u}$ que se define como

$$\nabla \mathbf{u} = \begin{bmatrix} \frac{\partial u_x}{\partial x} & \frac{\partial u_x}{\partial y} & \frac{\partial u_x}{\partial z} \\ \frac{\partial u_y}{\partial x} & \frac{\partial u_y}{\partial y} & \frac{\partial u_y}{\partial z} \\ \frac{\partial u_z}{\partial x} & \frac{\partial u_z}{\partial y} & \frac{\partial u_z}{\partial z} \end{bmatrix}. \tag{2.24}$$

Para su estudio se suele recurrir al cálculo de sus invariantes que se obtienen del polinomio característico, el cual se obtiene en cada punto del espacio como

$$\lambda^3 + I\lambda^2 + II\lambda + J = 0. \tag{2.25}$$

En flujos incompresibles los coeficientes del polinomio característico se pueden calcular a partir de las siguientes expresiones

$$
\begin{aligned}
I &= -\frac{\partial u_i}{\partial i} = 0 \quad \text{ya que} \quad \nabla \cdot \mathbf{u} = 0, \\
II &= -\frac{1}{2}\frac{\partial u_j}{\partial i}\frac{\partial u_i}{\partial j} = -\left(\frac{\partial u_y}{\partial x}\frac{\partial u_x}{\partial y} + \frac{\partial u_z}{\partial x}\frac{\partial u_x}{\partial z} + \frac{\partial u_z}{\partial y}\frac{\partial u_y}{\partial z}\right), \\
J &= -\frac{1}{3}\frac{\partial u_j}{\partial i}\frac{\partial u_k}{\partial j}\frac{\partial u_i}{\partial k} = \begin{vmatrix} \frac{\partial u_x}{\partial x} & \frac{\partial u_x}{\partial y} & \frac{\partial u_x}{\partial z} \\ \frac{\partial u_y}{\partial x} & \frac{\partial u_y}{\partial y} & \frac{\partial u_y}{\partial z} \\ \frac{\partial u_z}{\partial x} & \frac{\partial u_z}{\partial y} & \frac{\partial u_z}{\partial z} \end{vmatrix}.
\end{aligned}
\tag{2.26}
$$

Si los invariantes II y J en cada punto del espacio los representamos en un plano y trazamos isocontornos de probabilidad, obtenemos una representación gráfica de la distribución estadística de las variaciones del vector de velocidad como la de la figura 2.3. En el capítulo 7 se puede ver, aparte de la gráfica de isocontornos de los resultados que adopta la forma característica esperada [17], una gráfica en color que ofrece más precisión en su representación gráfica.

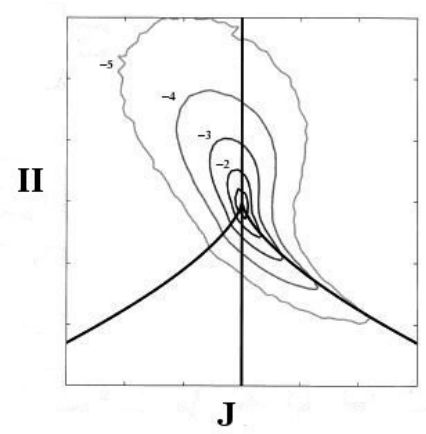

Figura 2.3: Forma típica de la gráfica de invariantes.

Capítulo 3

Análisis de Fourier

El método pseudoespectral que vamos a utilizar se basa en el uso de series de Fourier[†]. Debido a la imposibilidad de usar funciones continuas en una simulación numérica, en el modelo que se presenta se va a utilizar la transformada discreta de Fourier (DFT) [21][20] mediante la librería FFTW [8].

3.1. Series de Fourier

Las series de Fourier nos permiten aproximar una función periódica en función de unas sumas ponderadas de senos y cosenos. En este proyecto se tiene un campo de velocidades en un cubo de lado 2π que se repite periódicamente en el espacio y que puede aproximarse por una representación en series de Fourier. Para calcular los coeficientes de las series de Fourier se va a usar la transformada discreta de Fourier (DFT) para lo cual, en la sección 3.3, se establece la relación entre ambas.

Primero vamos a definir la serie de Fourier y sus propiedades en los casos unidimensionales para luego extender los resultados a nuestro problema, donde tenemos funciones vectoriales en tres dimensiones dependientes de tres variables.

Vamos a definir la representación en series de Fourier de un vector como la de cada una de sus componentes por separado. Así pues, en el siguiente apartado se define la serie de Fourier de una función dependiente de tres variables, que es una de las herramientas que se utilizan en la resolución del método pseudoespectral.

3.1.1. Definición

3.1.1.1. Series de Fourier en una dimensión

La definición de la serie de Fourier en una dimensión de una función $f : \mathbb{R} \to \mathbb{R}$ periódica en x de periodo X_0 es

$$\widehat{f}(x) = \sum_{k=-\infty}^{\infty} c_k e^{i\frac{2\pi}{X_0}kx}, \tag{3.1}$$

[†]Jean Baptiste Joseph Fourier (1768-1830), matemático y físico francés conocido por sus trabajos sobre la descomposición de funciones periódicas en series trigonométricas convergentes llamadas Series de Fourier. Se incorporó a la Escuela Normal Superior donde tuvo entre sus profesores a Joseph Louis Lagrange y Pierre Simon Laplace. Posteriormente, ocuparía una cátedra en la Escuela Politécnica, a partir de 1817 formó parte de la Academia de Ciencias Francesa. Como militar acompañó a Napoleón en su campaña por Egipto.

donde c_k son los coeficientes de la serie de Fourier y se definen como

$$c_k = \widehat{f}_k(x) = \frac{1}{X_0}\int_0^{X_0} f(x)e^{-i\frac{2\pi}{X_0}kx}dx. \tag{3.2}$$

Si se cumplen las condiciones de Dirichlet para las series de Fourier, es decir, que la función sea periódica y contínua salvo en un número finito de puntos y que tenga un número finito de máximos y mínimos locales estrictos en el periodo, la expresión 3.1 es la función f salvo en los puntos de discontinuidad (x_0) donde su valor es $\frac{f(x_0^+)+f(x_0^-)}{2}$.

3.1.1.2. Series de Fourier en tres dimensiones

La definición de la serie de Fourier en tres dimensiones de una función $f:\mathbb{R}^3\to\mathbb{R}$ que es periódica en las tres variables con el mismo periodo X_0 es

$$\begin{aligned}\widehat{f}(x_1,x_2,x_3) &= \sum_{k_1=-\infty}^{\infty}\sum_{k_2=-\infty}^{\infty}\sum_{k_3=-\infty}^{\infty} c_{k_1}c_{k_2}c_{k_3}e^{i\frac{2\pi}{X_0}k_1x_1}e^{i\frac{2\pi}{X_0}k_2x_2}e^{i\frac{2\pi}{X_0}k_3x_3}\\ &= \sum_{k_1=-\infty}^{\infty}\sum_{k_2=-\infty}^{\infty}\sum_{k_3=-\infty}^{\infty} c_{\mathbf{k}}e^{i\frac{2\pi}{X_0}\mathbf{k}\cdot\mathbf{x}},\end{aligned} \tag{3.3}$$

donde $c_{\mathbf{k}}$ son los coeficientes de la serie de Fourier y se definen como

$$c_{\mathbf{k}} = \widehat{f}_{\mathbf{k}} = \frac{1}{X_0^3}\int_0^{X_0}\int_0^{X_0}\int_0^{X_0} f(x_1,x_2,x_3)e^{-i\frac{2\pi}{X_0}(k_1x_1+k_2x_2+k_3x_3)}dx_1dx_2dx_3. \tag{3.4}$$

Esta aproximación es pues el resultado de aplicar las series de Fourier unidimensionales a una función que depende de tres variables tres veces consecutivas, transformando en cada una de ellas una de las variables del espacio físico al de Fourier.

3.1.2. Propiedades

En esta sección vamos a ver distintas propiedades que vamos a usar de las series de Fourier. En particular cómo se obtienen los coeficientes de las series de funciones que son el resultado de ciertas operaciones matemáticas.

3.1.2.1. Suma

Si queremos calcular los coeficientes de la serie de Fourier del resultado de la suma de dos funciones, $f(x)$ y $g(x)$, del mismo periodo X_0, donde

$$\widehat{f}(x) = \sum_{k=-\infty}^{\infty} d_k e^{i\frac{2\pi}{X_0}kx}, \tag{3.5}$$

$$\widehat{g}(x) = \sum_{k=-\infty}^{\infty} e_k e^{i\frac{2\pi}{X_0}kx}, \tag{3.6}$$

siendo d_k y e_k son los coeficientes de la series de Fourier de $f(x)$ y $g(x)$ respectivamente. El resultado de la suma es

$$\widehat{f}(x)+\widehat{g}(x) = \sum_{k=-\infty}^{\infty} d_k e^{i\frac{2\pi}{X_0}kx} + \sum_{k=-\infty}^{\infty} e_k e^{i\frac{2\pi}{X_0}kx} = \sum_{k=-\infty}^{\infty}(d_k+e_k)e^{i\frac{2\pi}{X_0}kx}. \tag{3.7}$$

Por tanto, el cálculo de los coeficientes de la serie de Fourier de la suma de funciones a partir de los coeficientes de las funciones por separado es

$$\widehat{(f+g)}(x) = \sum_{k=-\infty}^{\infty} c_k e^{i\frac{2\pi}{X_0}kx}, \tag{3.8}$$

donde

$$c_k = d_k + e_k. \tag{3.9}$$

Es decir, los coeficientes se calculan como la suma de los coeficientes de las funciones por separado. Es inmediato extender esta propiedad a tres dimensiones

$$c_{\mathbf{k}} = d_{\mathbf{k}} + e_{\mathbf{k}}. \tag{3.10}$$

3.1.2.2. Derivada espacial

Si queremos calcular los coeficientes de la serie de Fourier del resultado de la derivada respecto a la variable espacial de la función unidimensional $f(x)$ tenemos

$$d_k = \frac{1}{X_0}\int_0^{X_0} f'(x)e^{-i\frac{2\pi}{X_0}kx}dx. \tag{3.11}$$

Integrando por partes

$$d_k = \frac{1}{X_0}\left[f(x)e^{-i\frac{2\pi}{X_0}kx}\Big|_0^{X_0} - \int_0^{X_0}\left(-i\frac{2\pi}{X_0}k\right)f(x)e^{-i\frac{2\pi}{X_0}kx}dx\right]. \tag{3.12}$$

Como la función es periódica de periodo X_0 y para cualquier k se cumple que $e^{-i2\pi k} = 1$ desaparece el primer término, resultando

$$d_k = \frac{1}{X_0}\left[\left(i\frac{2\pi}{X_0}k\right)\int_0^{X_0} f(x)e^{-i\frac{2\pi}{X_0}kx}dx\right], \tag{3.13}$$

y por la definición 3.2

$$d_k = i\left(\frac{2\pi}{X_0}k\right)c_k. \tag{3.14}$$

Es sencillo extender esta propiedad a tres dimensiones a partir de la ecuación 3.4 con lo que se obtiene

$$\left(\widehat{\frac{\partial f_{\mathbf{k}}}{\partial x_l}}\right)_{\mathbf{k}} = i\left(\frac{2\pi}{X_0}k_l\right)c_{\mathbf{k}} \qquad \text{para} \quad l = 1,2,3. \tag{3.15}$$

En particular

$$\widehat{\nabla \mathbf{u}_{\mathbf{k}}} = i\frac{2\pi}{X_0}\mathbf{k}\cdot\widehat{\mathbf{u}}_{\mathbf{k}} \tag{3.16}$$

y

$$\widehat{\nabla^2 \mathbf{u}_{\mathbf{k}}} = -\left(\frac{2\pi}{X_0}\right)^2 k^2\widehat{\mathbf{u}}_{\mathbf{k}} \tag{3.17}$$

donde k^2 representa al módulo de $\mathbf{k}$ al cuadrado.

3.1.2.3. Derivada temporal. Derivada término a término

En los cálculos espectrales que se van a desarrollar en el capítulo 4 se realiza la evolución temporal de los coeficientes de Fourier. En la resolución se utiliza la siguiente propiedad

$$\widehat{\frac{\partial \mathbf{u}}{\partial t}} = \frac{\partial \widehat{\mathbf{u}}}{\partial t}. \tag{3.18}$$

Esta propiedad que no es cierta en general se cumple cuando las funciones verifican las condiciones de Dirichlet para las series de Fourier. El campo de velocidades que se usa en este proyecto tiene condiciones de contorno periódicas y cumple las condiciones de Dirichlet por lo que se puede aplicar esta propiedad.

3.2. Transformada Discreta de Fourier (DFT)

La DFT transforma una secuencia discreta de valores de una función en otra de la misma longitud que resume la influencia de cada frecuencia discreta en la secuencia original. Aplicada a nuestro problema, la DFT transforma una matriz tridimensional, que representa una variable fluida en el espacio físico, al espacio de Fourier, donde podemos obtener la información asociada a cada número de onda. En la sección 3.3 se establece como utilizarla para calcular los coeficientes de la serie de Fourier y, por tanto, utilizarla para los cálculos espectrales del método que se usa en este proyecto.

3.2.1. DFT en una dimensión

La transformada discreta de Fourier en una dimensión [21][20] de una función $f(n)$ de la que se toman N muestras en los puntos $n = 0, 1, \cdots, N-1$ se define como

$$F(k) = \frac{1}{N} \sum_{n=0}^{N-1} f(n) \cdot e^{-2\pi i k n/N} \qquad \text{donde } k = 0, 1, \cdots N-1. \tag{3.19}$$

Y su transformada inversa

$$f(n) = \sum_{k=0}^{N-1} F(k) \cdot e^{2\pi i k n/N} \qquad \text{donde } n = 0, 1, \cdots N-1. \tag{3.20}$$

Esta transformada permite obtener por ejemplo una representación del espectro de un campo de velocidades del que sólo conocemos los valores en determinados puntos del espacio. El espectro que obtenemos mediante la DFT se puede interpretar como un muestreo del espectro continuo tal y como puede verse en la ecuación 3.31. La influencia del número de muestras que tomamos en el cubo se puede resumir así: dado que el número de muestras es igual al número de frecuencias que obtenemos con esta transformada, y que éstas son los múltiplos de la frecuencia de muestreo, para analizar frecuencias más altas debemos tomar más muestras o tomarlas con una frecuencia de muestreo mayor, aunque esto último hace que las muestras en frecuencia estén más separadas entre sí.

A continuación vamos a ver un ejemplo que ilustra el uso de la DFT, para lo cual partimos de la siguiente función:

$$f(x) = \operatorname{sen}(30x) + 5\operatorname{sen}(4x) \qquad x \in [0, 2\pi). \tag{3.21}$$

Lo primero que se hace es tomar muestras equiespaciadas de la señal y construir la secuencia discreta $f(n)$ que se representa en la figura 3.1 donde

$$f(n) = f(x)|_{x=\frac{2\pi n}{N}} \qquad \text{donde } n = 1, 2, \cdots, N-1. \tag{3.22}$$

Tras aplicar la DFT a $f(n)$ podemos calcular la densidad espectral de potencia de la función definida como la suma al cuadrado de la parte real y la imaginaria de cada componente espectral. En la figura 3.2 se puede ver la densidad espectral de potencia del ejemplo, se observa que las únicas frecuencias que tienen un valor distinto de cero son las componentes asociadas a las frecuencias de la señal original.

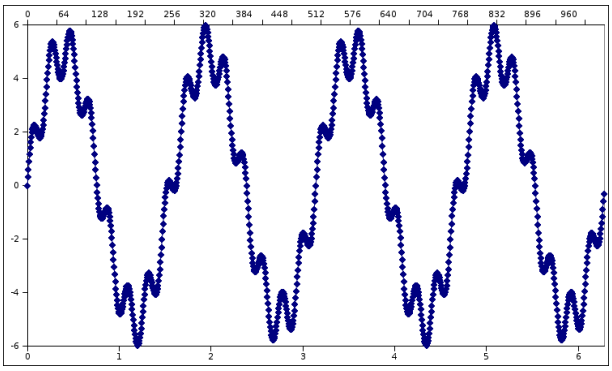

Figura 3.1: Representación de $f(n)$ con $N = 1024$ puntos. El eje X superior se corresponde con los valores de n y el inferior con los valores de $x = \frac{2\pi n}{N}$.

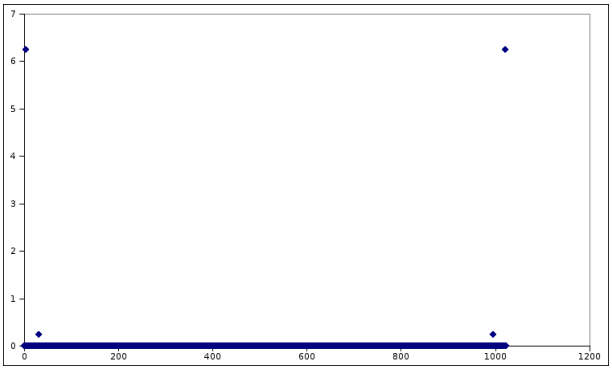

Figura 3.2: Espectro de $f(n)$.

3.2.2. DFT en tres dimensiones

La extensión a tres dimensiones de la DFT que vamos a usar consiste en tres DFT consecutivas una en cada dirección del espacio, por lo que la transformada queda:

$$\begin{aligned} F(k_1, k_2, k_3) &= DFT_{3D}\{f(n_1, n_2, n_3)\} \\ &= \frac{1}{N_1 N_2 N_3} \sum_{n_3=0}^{N_3-1} \sum_{n_2=0}^{N_2-1} \sum_{n_1=0}^{N_1-1} f(n_1, n_2, n_3) \cdot e^{-\frac{2\pi i k_1 n}{N_1}} \cdot e^{-\frac{2\pi i k_2 n}{N_2}} \cdot e^{-\frac{2\pi i k_3 n}{N_3}}. \end{aligned} \tag{3.23}$$

Y su transformada inversa es

$$\begin{aligned} f(n_1,n_2,n_3) &= DFT_{3D}\left\{F(k_1,k_2,k_3)\right\} \\ &= \sum_{k_3=0}^{N_3-1}\sum_{k_2=0}^{N_2-1}\sum_{k_1=0}^{N_1-1} F(k_1,k_2,k_3)\cdot e^{\frac{2\pi i k_1 n}{N_1}}\cdot e^{\frac{2\pi i k_2 n}{N_2}}\cdot e^{\frac{2\pi i k_3 n}{N_3}}. \end{aligned} \tag{3.24}$$

3.2.3. DFT de una secuencia real

En este proyecto se aplica la DFT a un campo de velocidades que sólo tiene componentes reales por lo que a continuación comentamos las características de este tipo de transformadas.

La DFT de una secuencia real cumple la siguiente propiedad de simetría en el espacio de Fourier:

$$F(k) = F^*(N-k). \tag{3.25}$$

A continuación comprobamos la validez de la misma:

$$\begin{aligned} F^*(N-k) &= \left(\frac{1}{N}\sum_{n=0}^{N-1} f(n)e^{-2\pi i n\frac{N-k}{N}}\right)^* = \left(\frac{1}{N}\sum_{n=0}^{N-1} f(n)e^{-2\pi i n}e^{2\pi i n\frac{k}{N}}\right)^* \\ &= \left(\frac{1}{N}\sum_{n=0}^{N-1} f(n)e^{2\pi i n\frac{k}{N}}\right)^* = \frac{1}{N}\sum_{n=0}^{N-1} f^*(n)e^{-2\pi i n\frac{k}{N}}. \end{aligned} \tag{3.26}$$

Si $f(n)$ es real entonces $f(n) = f^*(n)$ y se cumple la propiedad 3.25. Por tanto, sólo se necesita calcular la mitad de la DFT y la otra mitad se obtiene aplicando esta propiedad, reduciéndose así el coste computacional.

Otra ventaja de aplicar esta propiedad es el ahorro de memoria. Dado que a partir de la mitad de la DFT podemos obtener la otra mitad solamente necesitamos guardar una de las mitades. En nuestro caso transformamos un campo de velocidades (que son números reales). Como se realizan tres transformadas consecutivas cada una en una dirección espacial, en la primera transformada podemos aplicar esta propiedad; pero no para las otras dos ya que el resultado de la primera transformada es un campo de números complejos a los que no se les puede aplicar esta propiedad.

Por tanto, aplicando esta propiedad las necesidades de memoria tanto en espacio como en el dominio de Fourier son aproximadamente las mismas, dependiendo de si N es par o impar, ya que aunque en principio en el dominio de Fourier tenemos la mitad de los puntos, en cada uno de estos tenemos un número complejo.

3.2.4. La DFT en el problema de estudio

Vamos a aplicar la DFT a un cubo tridimensional de lado 2π extendido periódicamente por el espacio en las tres direcciones (ver figura 3.3).

Para poder usar la DFT en el problema hay que tener una representación de muestras discretas del cubo. Si nos fijamos en una dirección, como el lado del cubo es 2π, tenemos que discretizando la variable continua x obtenemos la variable discreta x_n de la siguiente forma:

$$x_n = \frac{2\pi n}{N} \quad \text{donde} \quad n = 0, 1, \cdots, N-1. \tag{3.27}$$

Es decir, se toman N muestras equiespaciadas en el rango $[0, 2\pi)$.

Un aspecto interesante de la DFT es que es una transformada lineal. En particular si $F(k)$ es la transformada de $f(x)$ entonces la transformada de $af(x)$ es $aF(k)$, por esta propiedad algunos autores definen la transformada con el factor $\frac{1}{N}$ en la transformada inversa.

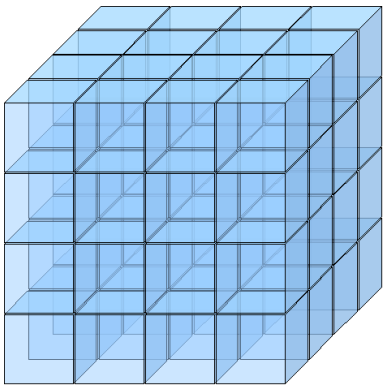

Figura 3.3: Extensión periódica del cubo de lado 2π.

En nuestro caso, después de analizar el sistema, vemos que resulta más ventajoso utilizar las expresiones de las ecuaciones 3.19 y 3.20 ya que en cada paso integral se realizan 3 transformadas directas y 6 transformadas inversas. Además en el cálculo de invariantes se realizan otras 9 transformadas inversas por lo que eligiendo adecuadamente dónde escalar ahorramos un gran coste computacional.

3.2.5. FFT

La transformada DFT expresada en la ecuación 3.19 requiere de $2N^2$ operaciones computacionales, esta cifra se puede reducir bastante usando las propiedades de la misma hasta $2N \log N$. Esta reducción se conoce como el algoritmo FFT[†] (Fast Fourier Transform) [20][18]. En la figura 3.4 se puede comprobar como cuanto mayor es el número de muestras mayor es la eficiencia de la FFT. Para poder reducir el número de operaciones lo que se hace es agrupar el conjunto de muestras de una forma determinada y aplicar la DFT a esos subconjuntos.

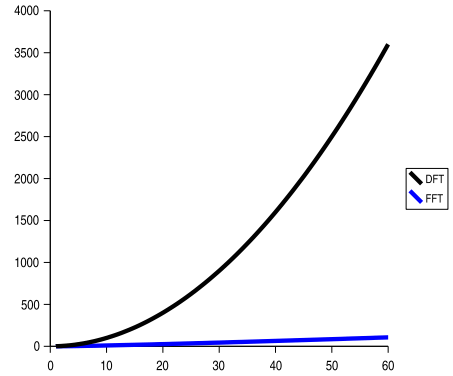

Figura 3.4: Comparativa entre el coste computacional de la DFT y la FFT.

La librería FFTW

En el programa vamos a usar la librería fftw que según su documentación[‡] realiza los cálculos indicados en la ecuación 3.28 para la transformada directa donde si denominamos

[†]El algoritmo FFT fue desarrollado por James Cooley y John Tukey en 1965.

[‡]En esta sección se usa la notación de la documentación de la librería FFTW si bien es sencilla su extrapolación a la del resto del proyecto.

s a una de las dimensiones, definimos $\omega_s = e^{\frac{2\pi}{n_s}i}$ siendo n_s el número de elementos en la dimensión s.

$$Y[i_1, i_2, \cdots, i_d] = \sum_{j_1=0}^{n_1-1} \sum_{j_2=0}^{n_2-1} \cdots \sum_{j_n=0}^{n_n-1} X[j_1, j_2, \cdots, j_d] \omega_1^{-i_1 j_1} \omega_2^{-i_2 j_2} \cdots \omega_n^{-i_n j_n}. \tag{3.28}$$

La transformada inversa realiza los siguientes cálculos:

$$Y[i_1, i_2, \cdots, i_d] = \sum_{j_1=0}^{n_1-1} \sum_{j_2=0}^{n_2-1} \cdots \sum_{j_n=0}^{n_n-1} X[j_1, j_2, \cdots, j_d] \omega_1^{i_1 j_1} \omega_2^{i_2 j_2} \cdots \omega_n^{i_n j_n}. \tag{3.29}$$

Comparando con 3.23 y 3.24 vemos que en la transformada 3.28 falta el término $\frac{1}{N_1 N_2 N_3}$. Es decir que la librería calcula la transformada desnormalizada por lo que después de hacer la transformada directa hay que multiplicar todos los términos por $\frac{1}{N_1 N_2 N_3}$.

3.3. Relación de la DFT con la serie de Fourier

En esta sección se analiza el uso de la DFT para realizar los cálculos de los coeficientes de Fourier. Si partimos de una función f periódica de periodo X_0 que verifica las condiciones de Dirichlet, se puede representar con una serie de Fourier de la forma de la ecuación 3.1 donde c_k son los coeficientes de la serie de Fourier cuya expresión recordamos a continuacion

$$c_k = \widehat{f_k} = \frac{1}{X_0} \int_0^{X_0} f(x) e^{-i\frac{2\pi}{X_0}kx} dx.$$

Para estimar estos coeficientes aplicamos la regla del trapecio. Dividimos el intervalo $[0, X_0]$ en N subintervalos del mismo tamaño con extremos en los puntos $x_n = \frac{nX_0}{N}$, entonces usando que $f(0) = f(X_0)$ por la periodicidad obtenemos

$$\begin{aligned} c_k &= \frac{1}{X_0} \sum_{n=0}^{N-1} \int_{n\frac{X_0}{N}}^{(n+1)\frac{X_0}{N}} f(x) e^{-\frac{i2\pi}{X_0}kx} dx \\ &\simeq \frac{1}{X_0} \sum_{n=0}^{N-1} \frac{1}{2} \left[f(n\frac{X_0}{N}) e^{-\frac{i2\pi}{X_0}kn\frac{X_0}{N}} + f((n+1)\frac{X_0}{N}) e^{-\frac{i2\pi}{X_0}k(n+1)\frac{X_0}{N}} \right] \frac{X_0}{N} \\ &= \frac{1}{X_0}\frac{X_0}{N} \left(\frac{f(0)}{2} + \sum_{n=1}^{N-1} f(x_n) e^{-i2\pi kn/N} + \frac{f(X_0)}{2} \right) = \frac{1}{N} \sum_{n=0}^{N-1} f(x_k) e^{-i2\pi kn/N}. \end{aligned} \tag{3.30}$$

Comparando con 3.19 vemos que

$$(c_0, c_1, \cdots, c_{N-1}) \simeq DFT\left(f(x_0), f(x_1), \cdots, f(x_{N-1})\right). \tag{3.31}$$

Es decir, con la definición de DFT que se usa en este proyecto los valores en frecuencia se aproximan a los coeficientes de la serie de Fourier de la función.

Capítulo 4

Desarrollo teórico

4.1. Introducción

Para el cálculo de la evolución temporal de un flujo turbulento homogéneo e isótropo, vamos a recurrir a las ecuaciones de Navier[†]-Stokes[‡]. En este capítulo se desarrolla el proceso matemático que conduce a la ecuación que utilizaremos en el programa para calcular el estado del campo de velocidades en un paso de integración a partir de los anteriores.

Partiremos de las ecuaciones de Navier-Stokes para un flujo incompresible

$$\frac{\partial \mathbf{u}}{\partial t} = \mathbf{u} \times \omega - \nabla \left(\frac{p}{\rho} + \frac{1}{2} u^2 \right) + \nu \nabla^2 \mathbf{u} + \mathbf{f}, \tag{4.1}$$

$$\nabla \cdot \mathbf{u} = 0. \tag{4.2}$$

Donde $\mathbf{u}$ es el vector de velocidad, ω es el rotacional de $\mathbf{u}$, p es la presión, ρ es la densidad del fluido y ν es su viscosidad cinemática. Además se ha añadido un término fuente de cantidad de movimiento $\mathbf{f}$ necesario para conseguir un flujo turbulento estadísticamente estacionario.

4.2. Aproximación por series de Fourier

Vamos a aproximar cada término de las ecuaciones de Navier-Stokes por una serie de Fourier. Posteriormente obtendremos una simplificación que nos permitirá eliminar la depen-

[†]Louis-Marie Henri Navier (1785-1836) matemático e ingeniero francés que contribuyó notablemente al desarrollo de la hidrodinámica. En 1808 ingresó en el cuerpo de Ingenieros de Puentes y Calzadas, de cuya academia fue nombrado profesor en 1819. En 1824 ingresó en la prestigiosa Academia de Ciencias, dónde tuvo una prolífica carrera investigadora. En 1823 publicó su trabajo esencial, *Sur les mouvements des fluides en ayant égard à l'adhésion des molécules*, en el que partiendo de un modelo molecular dedujo las importantes ecuaciones que describen el comportamiento de los fluidos llamados reales o viscosos (en contraposición a los ideales).

[‡]Sir George Gabriel Stokes (1819-1903) matemático y físico británico, nacido en Skreen. Estudió numerosos temas de hidrodinámica estableciendo la teoría de la viscosidad de los fluidos y formulando la ley que lleva su nombre. Esta ley permite calcular el movimiento de pequeñas esferas en el seno de medios con elevada viscosidad. Contribuyó a la teoría ondulatoria de la luz con su ley sobre la refrangibilidad variable de la luz. Realizó el estudio de las radiaciones ultravioleta mediante los fenómenos de fluorescencia a que dan lugar. En 1896 propuso la idea de que los rayos X, recién descubiertos por Röntgen, eran radiaciones de tipo electromagnético.

dencia con $\frac{p}{\rho}$. Aplicando la propiedad 3.18 sobre la ecuación 4.1 obtenemos

$$\frac{\partial \widehat{\mathbf{u}}}{\partial t} = \widehat{\frac{\partial \mathbf{u}}{\partial t}} = \overbrace{\left(\overbrace{\mathbf{u} \times \omega}\right)} - \overbrace{\left[\overbrace{\nabla\left(\frac{p}{\rho} + \frac{1}{2}u^2\right)}\right]} + \overbrace{\left(\overbrace{\nu\nabla^2\mathbf{u}}\right)} + \widehat{\mathbf{f}}. \tag{4.3}$$

Vamos a ver la evolución temporal de cada coeficiente de las series. Fijando $\mathbf{k}$ y aplicando la propiedad 3.10, obtenemos

$$\frac{\partial \widehat{\mathbf{u}}_\mathbf{k}}{\partial t} = \overbrace{\left(\overbrace{\mathbf{u} \times \omega}\right)}_\mathbf{k} - \overbrace{\left[\overbrace{\nabla\left(\frac{p}{\rho} + \frac{1}{2}u^2\right)}\right]}_\mathbf{k} + \overbrace{\left(\overbrace{\nu\nabla^2\mathbf{u}}\right)}_\mathbf{k} + \widehat{\mathbf{f}}_\mathbf{k}. \tag{4.4}$$

Reordenando la ecuación y aplicando la propiedad 3.15 se obtiene

$$\frac{\partial \widehat{\mathbf{u}}_\mathbf{k}}{\partial t} = \overbrace{\left(\overbrace{\mathbf{u} \times \omega}\right)}_\mathbf{k} + \widehat{\mathbf{f}}_\mathbf{k} - \left[i\mathbf{k}\overbrace{\left(\frac{p}{\rho} + \frac{1}{2}u^2\right)}_\mathbf{k}\right] - \nu k^2 \widehat{\mathbf{u}}_\mathbf{k}. \tag{4.5}$$

A continuación se va a aplicar una simplificación que nos permite eliminar la dependencia con $\frac{p}{\rho}$. Aplicando la divergencia a la ecuación 4.1, tenemos que la derivada temporal se anula debido a la ecuación 4.2 y queda

$$0 = \nabla \cdot (\mathbf{u} \times \omega) - \nabla^2\left(\frac{p}{\rho} + \frac{1}{2}u^2\right) + \nabla \cdot \mathbf{f}. \tag{4.6}$$

Aproximando con series de Fourier, quedándonos con el k-ésimo coeficiente y aplicando la propiedad 3.15 obtenemos

$$0 = i\,\mathbf{k} \cdot \overbrace{\left(\overbrace{\mathbf{u} \times \omega}\right)}_\mathbf{k} - i\,\mathbf{k} \cdot \left[\overbrace{\nabla\left(\frac{p}{\rho} + \frac{1}{2}u^2\right)}\right]_\mathbf{k} + i\,\mathbf{k} \cdot \widehat{\mathbf{f}}_\mathbf{k}. \tag{4.7}$$

Simplificando y reordenando los términos

$$\mathbf{k} \cdot \left[\overbrace{\left(\overbrace{\mathbf{u} \times \omega}\right)}_\mathbf{k} + \widehat{\mathbf{f}}_\mathbf{k}\right] = \mathbf{k} \cdot \left[\overbrace{\nabla\left(\frac{p}{\rho} + \frac{1}{2}u^2\right)}\right]_\mathbf{k}. \tag{4.8}$$

A continuación volvemos a aplicar la propiedad 3.15 al término de la derecha de la igualdad

$$\mathbf{k} \cdot \left[\overbrace{\left(\overbrace{\mathbf{u} \times \omega}\right)}_\mathbf{k} + \widehat{\mathbf{f}}_\mathbf{k}\right] = \mathbf{k} \cdot \left[\mathbf{k}\overbrace{\left(\frac{p}{\rho} + \frac{1}{2}u^2\right)}_\mathbf{k}\right] i, \tag{4.9}$$

desarrollando y operando con los productos escalares obtenemos

$$\sum_{i=1}^{3} k_i \left[\overbrace{\left(\overbrace{\mathbf{u} \times \omega}\right)}_{k_i} + \widehat{\mathbf{f}}_{k_i}\right] = k^2 \overbrace{\left(\frac{p}{\rho} + \frac{1}{2}u^2\right)}_\mathbf{k} i, \tag{4.10}$$

$$\sum_{i=1}^{3} \frac{k_i}{k^2} \left[\overbrace{\left(\overbrace{\mathbf{u} \times \omega}\right)}_{k_i} + \widehat{\mathbf{f}}_{k_i}\right] = \overbrace{\left(\frac{p}{\rho} + \frac{1}{2}u^2\right)}_\mathbf{k} i. \tag{4.11}$$

Esta ecuación se puede sustituir en la 4.5

$$\frac{\partial \widehat{\mathbf{u}}_{\mathbf{k}}}{\partial t} = \left(\widehat{\mathbf{u}\times\omega}\right)_{\mathbf{k}} + \widehat{\mathbf{f}}_{\mathbf{k}} - \mathbf{k}\left\{\sum_{i=1}^{3}\frac{k_i}{k^2}\left[\left(\widehat{\mathbf{u}\times\omega}\right)_{k_i} + \widehat{\mathbf{f}}_{k_i}\right]\right\} - \nu k^2\widehat{\mathbf{u}}_{\mathbf{k}}. \tag{4.12}$$

Tenemos una ecuación vectorial que expresada en una de sus componentes resulta

$$\frac{\partial \widehat{\mathbf{u}}_{k_j}}{\partial t} = \left(\widehat{\mathbf{u}\times\omega}\right)_{k_j} + \widehat{\mathbf{f}}_{k_j} - k_j\left\{\sum_{i=1}^{3}\frac{k_i}{k^2}\left[\left(\widehat{\mathbf{u}\times\omega}\right)_{k_i} + \widehat{\mathbf{f}}_{k_i}\right]\right\} - \nu k^2\widehat{\mathbf{u}}_{k_j}. \tag{4.13}$$

Para poder sacar factor común introducimos una delta de Kronecker

$$\frac{\partial \widehat{\mathbf{u}}_{k_j}}{\partial t} = \sum_{i=1}^{3}\delta_{ij}\left[\left(\widehat{\mathbf{u}\times\omega}\right)_{k_i} + \widehat{\mathbf{f}}_{k_i}\right] - \sum_{i=1}^{3}\frac{k_ik_j}{k^2}\left[\left(\widehat{\mathbf{u}\times\omega}\right)_{k_i} + \widehat{\mathbf{f}}_{k_i}\right] - \nu k^2\widehat{\mathbf{u}}_{k_j}. \tag{4.14}$$

Se saca factor común

$$\frac{\partial \widehat{\mathbf{u}}_{k_j}}{\partial t} = \sum_{i=1}^{3}\left(\delta_{ij} - \frac{k_jk_i}{k^2}\right)\left[\left(\widehat{\mathbf{u}\times\omega}\right)_{k_i} + \widehat{\mathbf{f}}_{k_i}\right] - \nu k^2\mathbf{u}_{k_j}. \tag{4.15}$$

Para simplificar la expresión se puede definir la matriz $\mathbf{P} \in \mathbb{R}^3 \times \mathbb{R}^3$ de componentes

$$P_{ij} = \delta_{ij} - \frac{k_ik_j}{k^2} \tag{4.16}$$

y sustituyéndolos en la ecuación 4.15 llegamos a

$$\frac{\partial \widehat{\mathbf{u}}_{k_j}}{\partial t} = \sum_{i=1}^{3}P_{ij}\left[\left(\widehat{\mathbf{u}\times\omega}\right)_{k_i} + \widehat{\mathbf{f}}_{k_i}\right] - \nu k^2\mathbf{u}_{k_j}. \tag{4.17}$$

Finalmente expresando la ecuación (4.17) en forma vectorial queda

$$\frac{\partial \widehat{\mathbf{u}}_{\mathbf{k}}}{\partial t} = \mathbf{P}\left[\left(\widehat{\mathbf{u}\times\omega}\right)_{\mathbf{k}} + \widehat{\mathbf{f}}_{\mathbf{k}}\right] - \nu k^2\widehat{\mathbf{u}}_{\mathbf{k}}. \tag{4.18}$$

Y reordenándola llegamos a

$$\frac{\partial \widehat{\mathbf{u}}_{\mathbf{k}}}{\partial t} + \nu k^2\widehat{\mathbf{u}}_{\mathbf{k}} = \mathbf{P}\left[\left(\widehat{\mathbf{u}\times\omega}\right)_{\mathbf{k}} + \widehat{\mathbf{f}}_{\mathbf{k}}\right], \tag{4.19}$$

que es la ecuación que tenemos que integrar en el programa.

4.2.1. Cálculo del término no lineal

El término $\left(\widehat{\mathbf{u}\times\omega}\right)_k$ es el coeficiente de la serie de Fourier del resultado de realizar la multiplicación vectorial de la velocidad por la vorticidad. También se podrían haber calculado los coeficientes de las series de Fourier de la velocidad y la vorticidad por separado y operar completamente en frecuencia pero en ese caso el coste computacional sería mayor. En este apartado vamos a detallar los pasos que vamos a seguir en el programa para calcular este término. La definición de la vorticidad es:

$$\omega = \nabla \times \mathbf{u}. \tag{4.20}$$

Desarrollando el producto vectorial tenemos

$$\omega = \begin{vmatrix} \mathbf{e}_x & \mathbf{e}_y & \mathbf{e}_z \\ \frac{\partial}{\partial x} & \frac{\partial}{\partial y} & \frac{\partial}{\partial z} \\ u_x & u_y & u_z \end{vmatrix} = \left(\frac{\partial u_z}{\partial y} - \frac{\partial u_y}{\partial z}\right)\mathbf{e}_x + \left(\frac{\partial u_x}{\partial z} - \frac{\partial u_z}{\partial x}\right)\mathbf{e}_y + \left(\frac{\partial u_y}{\partial x} - \frac{\partial u_x}{\partial y}\right)\mathbf{e}_z. \quad (4.21)$$

El programa parte de los coeficientes de la serie de Fourier del vector de velocidad, por tanto para calcular las derivadas usamos la propiedad 3.15:

$$\omega_x = \frac{\partial u_z}{\partial y} - \frac{\partial u_y}{\partial z} \Longrightarrow \widehat{\omega}_x = ik_y\widehat{u}_z - ik_z\widehat{u}_y, \quad (4.22)$$

$$\omega_y = \frac{\partial u_x}{\partial z} - \frac{\partial u_z}{\partial x} \Longrightarrow \widehat{\omega}_y = ik_z\widehat{u}_x - ik_x\widehat{u}_z, \quad (4.23)$$

$$\omega_z = \frac{\partial u_y}{\partial x} - \frac{\partial u_x}{\partial y} \Longrightarrow \widehat{\omega}_z = ik_x\widehat{u}_y - ik_y\widehat{u}_x. \quad (4.24)$$

Con los coeficientes de la serie de Fourier de ω podemos obtener los valores de la función en los puntos del cubo en el espacio físico, y en ellos realizar los cálculos de multiplicación vectorial por la velocidad:

$$\mathbf{u}\times\omega = \begin{vmatrix} \mathbf{e}_x & \mathbf{e}_y & \mathbf{e}_z \\ u_x & u_y & u_z \\ \omega_x & \omega_y & \omega_z \end{vmatrix} = (u_y\omega_z - u_z\omega_y)\,\mathbf{e}_x + (u_z\omega_x - u_x\omega_z)\,\mathbf{e}_y + (u_x\omega_y - u_y\omega_x)\,\mathbf{e}_z. \quad (4.25)$$

Por último calculamos los coeficientes de la serie de Fourier del resultado que es lo que necesitamos para nuestros cálculos

$$\left(\overbrace{\mathbf{u}\times\omega}\right)_k = \left\{\overbrace{(u_y\omega_z - u_z\omega_y)}\right\}_k \mathbf{e}_x + \left\{\overbrace{(u_z\omega_x - u_x\omega_z)}\right\}_k \mathbf{e}_y + \left\{\overbrace{(u_x\omega_y - u_y\omega_x)}\right\}_k \mathbf{e}_z. \quad (4.26)$$

4.3. Integración numérica

4.3.1. Paso previo

Antes de integrar la ecuación 4.19 vamos a calcular la siguiente derivada que nos será de gran utilidad

$$\frac{d}{dt}\left(\widehat{\mathbf{u}}_\mathbf{k}e^{\nu k^2 t}\right) = \frac{d\widehat{\mathbf{u}}_\mathbf{k}}{dt}e^{\nu k^2 t} + \nu k^2\widehat{\mathbf{u}}_\mathbf{k}e^{\nu k^2 t} = e^{\nu k^2 t}\left(\frac{d\widehat{\mathbf{u}}_\mathbf{k}}{dt} + \nu k^2\widehat{\mathbf{u}}_\mathbf{k}\right). \quad (4.27)$$

Operando

$$\frac{d\widehat{\mathbf{u}}_\mathbf{k}}{dt} + \nu k^2\widehat{\mathbf{u}}_\mathbf{k} = \frac{d}{dt}\left(\widehat{\mathbf{u}}_\mathbf{k}e^{\nu k^2 t}\right)e^{-\nu k^2 t}. \quad (4.28)$$

Hemos obtenido una ecuación que podemos sustituir en la ecuación 4.19 para simplificar su cálculo con lo que se obtiene

$$\frac{d}{dt}\left(\widehat{\mathbf{u}}_\mathbf{k}e^{\nu k^2 t}\right)e^{-\nu k^2 t} = \mathbf{P}\left[\left(\overbrace{\mathbf{u}\times\omega}\right)_\mathbf{k} + \widehat{\mathbf{f}_\mathbf{k}}\right]. \quad (4.29)$$

Que se puede reescribir

$$\frac{d}{dt}\left(\widehat{\mathbf{u}}_\mathbf{k}e^{\nu k^2 t}\right) = e^{\nu k^2 t}\mathbf{P}\left[\left(\overbrace{\mathbf{u}\times\omega}\right)_\mathbf{k} + \mathbf{f}_\mathbf{k}\right]. \quad (4.30)$$

En los siguientes apartados presentamos los métodos de integración numérica, que vamos a utilizar en el programa, para resolver la ecuación 4.30. La elección de estos métodos se basa en la comparativa del artículo de Dale R. Durran [7].

4.3.2. Método de Euler

Como ejemplo vamos a desarrollar el método de Euler si bien en el código no se usa, ya que es un método de primer orden. El método parte de la siguiente ecuación diferencial

$$\frac{dy}{dt} = F(y,t),$$

y nos permite aproximar de forma numérica el valor del siguiente paso de integración mediante la siguiente ecuación en diferencias

$$y^{t_{n+1}} = y^{t_n} + dtF^{t_n}, \tag{4.31}$$

donde dt es el valor del paso de integración e y^{t_n} es la aproximación de y en el instante t_n previamente obtenida. Discretizando la ecuación 4.30 y aplicando la 4.31 donde $y = \mathbf{u_k}e^{\nu k^2 t}$ obtenemos

$$\widehat{\mathbf{u}}_{\mathbf{k}}^{t_{n+1}} e^{\nu k^2 (t_n+dt)} = \widehat{\mathbf{u}}_{\mathbf{k}}^{t_n} e^{\nu k^2 t_n} + dt \left\{ e^{\nu k^2 t_n} \mathbf{P} \left[\left(\widetilde{\mathbf{u} \times \omega} \right)_{\mathbf{k}}^{t_n} + \widehat{\mathbf{f}}_{\mathbf{k}}^{t_n} \right] \right\}. \tag{4.32}$$

Multiplicando por $e^{-\nu k^2 (t_n+dt)}$, la ecuación 4.32 queda

$$\widehat{\mathbf{u}}_{\mathbf{k}}^{t_{n+1}} = \widehat{\mathbf{u}}_{\mathbf{k}}^{t_n} e^{-\nu k^2 dt} + dt \left\{ e^{-\nu k^2 dt} \mathbf{P} \left[\left(\widetilde{\mathbf{u} \times \omega} \right)_{\mathbf{k}}^{t_n} + \widehat{\mathbf{f}}_{\mathbf{k}}^{t_n} \right] \right\}. \tag{4.33}$$

En nuestro problema el término de forzado es $\widehat{\mathbf{f}}_{\mathbf{k}}^{t_n} = 0$ y, en su lugar, la energía se inyecta al sistema introduciendo el concepto de viscosidad negativa en los modos mayores del campo de velocidades donde $|k| < 2{,}5$. En estos se sustituye ν por una cierta cantidad negativa. Esa cantidad negativa (que en el programa está representada por la variable *VISCOSIDADNEG*) es la llamada viscosidad negativa. Su valor varía en cada iteración controlada por un PID de manera que el fluido alcance un cierto grado de estabilidad (ver apartado 7.3), concretamente que la cantidad $KETA = k_{max}\eta$ fluctúe alrededor de un cierto valor constante, usualmente $KETA\_ESTACIONARIA = 1{,}4$.

Este valor está relacionado con el tamaño del rango disipativo. Algunos autores lo toman como 2, si bien, debido a que el espectro en el rango disipativo decrece exponencialmente (ver ecuación 2.22), el error cometido al tomarlo como 1,4 es despreciable y el coste computacional disminuye sensiblemente. Otra interpretación de este valor sería verlo a través de la separación entre las repeticiones del espectro que introduce la DFT, según el teorema de Nyquist es necesario utilizar como frecuencia de muestreo como mínimo el doble de la máxima frecuencia que se simula, pero como se conoce la forma del espectro se permite cierta intersección en la zona en la que su valor es prácticamente cero.

Finalmente simplificando la ecuación 4.33 obtenemos

$$\widehat{\mathbf{u}}_{\mathbf{k}}^{t_{n+1}} = \widehat{\mathbf{u}}_{\mathbf{k}}^{t_n} e^{-\nu k^2 dt} + dt \left\{ e^{-\nu k^2 dt} \mathbf{P} \left(\widetilde{\mathbf{u} \times \omega} \right)_{\mathbf{k}}^{t_n} \right\}, \tag{4.34}$$

$$\widehat{\mathbf{u}}_{\mathbf{k}}^{t_{n+1}} = e^{-\nu k^2 dt} \left\{ \widehat{\mathbf{u}}_{\mathbf{k}}^{t_n} + dt \mathbf{P} \left(\widetilde{\mathbf{u} \times \omega} \right)_{\mathbf{k}}^{t_n} \right\}. \tag{4.35}$$

4.3.3. Método de Runge-Kutta de segundo orden

Los primeros pasos de integración y los cambios en el tamaño del paso se hacen con el método de Runge-Kutta de segundo orden cuya ecuación en diferencias es

$$\begin{aligned} y^{t_{n+1}} &= y^{t_n} + dtF^{t_n} + \frac{dt}{2}\left[F\left(y_1\right) - F^{t_n}\right] \\ &= y^{t_n} + \frac{dt}{2}\left[F(y^{t_n}) + F\left(y_1\right)\right]. \end{aligned} \tag{4.36}$$

donde

$$y_1 = y^{t_n} + dtF^{t_n}. \tag{4.37}$$

Discretizando la ecuación 4.30 y aplicando la 4.36 donde $y = \mathbf{u_k}e^{\nu k^2 t}$ obtenemos

$$\begin{aligned} \widehat{\mathbf{u}}_{\mathbf{k}}^{t_{n+1}} e^{\nu k^2 (t_n + dt)} =& \widehat{\mathbf{u}}_{\mathbf{k}}^{t_n} e^{\nu k^2 t_n} + \frac{dt}{2}\left\{\left[e^{\nu k^2 t_n}\mathbf{P}\left(\left(\widehat{\mathbf{u}\times\omega}\right)_{\mathbf{k}}^{t_n} + \widehat{\mathbf{f}}_{\mathbf{k}}^{t_n}\right)\right] + \right. \\ & \left.\left[e^{\nu k^2 t_n}\mathbf{P}\left(\left(\widehat{\mathbf{u_1}\times\omega_1}\right)_{\mathbf{k}} + \widehat{\mathbf{f}}_{\mathbf{1_k}}\right)\right]\right\} \end{aligned} \tag{4.38}$$

donde

$$\widehat{\mathbf{u_{1k}}} = \widehat{\mathbf{u}}_{\mathbf{k}}^{t_n} + dt\mathbf{P}\left[\left(\widehat{\mathbf{u}\times\omega}\right)_{\mathbf{k}}^{t_n} + \widehat{\mathbf{f}}_{\mathbf{k}}^{t_n}\right]. \tag{4.39}$$

Multiplicando por la exponencial $e^{-\nu k^2(t_n+dt)}$, la ecuación 4.38 queda

$$\begin{aligned} \widehat{\mathbf{u}}_{\mathbf{k}}^{t_{n+1}} =& \widehat{\mathbf{u}}_{\mathbf{k}}^{t_n} e^{-\nu k^2 dt} + \frac{dt}{2}\left\{\left[e^{-\nu k^2 dt}\mathbf{P}\left(\left(\widehat{\mathbf{u}\times\omega}\right)_{\mathbf{k}}^{t_n} + \widehat{\mathbf{f}}_{\mathbf{k}}^{t_n}\right)\right] + \right. \\ & \left.\left[e^{-\nu k^2 dt}\mathbf{P}\left(\left(\widehat{\mathbf{u_1}\times\omega_1}\right)_{\mathbf{k}} + \widehat{\mathbf{f}}_{\mathbf{1_k}}\right)\right]\right\} \end{aligned} \tag{4.40}$$

Como en el apartado anterior, el término de forzado se anula, por tanto, simplificando la ecuación 4.40 tenemos

$$\widehat{\mathbf{u}}_{\mathbf{k}}^{t_{n+1}} = \widehat{\mathbf{u}}_{\mathbf{k}}^{t_n} e^{-\nu k^2 dt} + \frac{dt}{2}\left\{\left[e^{-\nu k^2 dt}\mathbf{P}\left(\left(\widehat{\mathbf{u}\times\omega}\right)_{\mathbf{k}}^{t_n}\right)\right] + \left[e^{-\nu k^2 dt}\mathbf{P}\left(\widehat{\mathbf{u_1}\times\omega_1}\right)_{\mathbf{k}}\right]\right\}, \tag{4.41}$$

$$\widehat{\mathbf{u}}_{\mathbf{k}}^{t_{n+1}} = e^{-\nu k^2 dt}\left\{\widehat{\mathbf{u}}_{\mathbf{k}}^{t_n} + \frac{dt}{2}\mathbf{P}\left[\left(\widehat{\mathbf{u}\times\omega}\right)_{\mathbf{k}}^{t_n} + \left(\widehat{\mathbf{u_1}\times\omega_1}\right)_{\mathbf{k}}\right]\right\}. \tag{4.42}$$

4.3.4. Método de Adams-Bashforth de segundo orden

El resto de los pasos de integración se hacen con el método de Adams-Bashforth de segundo orden que nos permite reducir considerablemente el coste computacional frente al anterior. La ecuación en diferencia de este método es

$$y^{t_{n+1}} = y^{t_n} + \frac{dt}{2}\left(3F^{t_n} - F^{t_{n-1}}\right). \tag{4.43}$$

Discretizando la ecuación 4.30 y aplicando la 4.43 donde $y = \mathbf{u_k} e^{\nu k^2 t}$ obtenemos

$$\begin{aligned}\widehat{\mathbf{u}}_{\mathbf{k}}^{t_{n+1}} e^{\nu k^2 (t_n + dt)} = \widehat{\mathbf{u}}_{\mathbf{k}}^{t_n} e^{\nu k^2 t_n} + \frac{dt}{2} \Bigg\{ 3 \left[e^{\nu k^2 t_n} \mathbf{P} \left(\left(\widetilde{\mathbf{u} \times \omega} \right)_{\mathbf{k}}^{t_n} + \widehat{\mathbf{f}}_{\mathbf{k}}^{t_n} \right) \right] - \\ \left[e^{\nu k^2 (t_n - dt)} \mathbf{P} \left(\left(\widetilde{\mathbf{u} \times \omega} \right)_{\mathbf{k}}^{t_{n-1}} + \widehat{\mathbf{f}}_{\mathbf{k}}^{t_{n-1}} \right) \right] \Bigg\}.\end{aligned} \tag{4.44}$$

Multiplicando por la exponencial $e^{-\nu k^2 (t_n + dt)}$, la ecuación 4.44 queda

$$\begin{aligned}\widehat{\mathbf{u}}_{\mathbf{k}}^{t_{n+1}} = \widehat{\mathbf{u}}_{\mathbf{k}}^{t_n} e^{-\nu k^2 dt} + \frac{dt}{2} \Bigg\{ 3 \left[e^{-\nu k^2 dt} \mathbf{P} \left(\left(\widetilde{\mathbf{u} \times \omega} \right)_{\mathbf{k}}^{t_n} + \widehat{\mathbf{f}}_{\mathbf{k}}^{t_n} \right) \right] - \\ \left[e^{-\nu k^2 (2dt)} \mathbf{P} \left(\left(\widetilde{\mathbf{u} \times \omega} \right)_{\mathbf{k}}^{t_{n-1}} + \widehat{\mathbf{f}}_{\mathbf{k}}^{t_{n-1}} \right) \right] \Bigg\}.\end{aligned} \tag{4.45}$$

Como en los apartados anteriores, el término de forzado se anula, por tanto, simplificando la ecuación 4.45 tenemos

$$\begin{aligned}\widehat{\mathbf{u}}_{\mathbf{k}}^{t_{n+1}} = \widehat{\mathbf{u}}_{\mathbf{k}}^{t_n} e^{-\nu k^2 dt} + \frac{dt}{2} \Bigg\{ 3 \left[e^{-\nu k^2 dt} \mathbf{P} \left(\widetilde{\mathbf{u} \times \omega} \right)_{\mathbf{k}}^{t_n} \right] - \\ \left[e^{-\nu k^2 (2dt)} \mathbf{P} \left(\widetilde{\mathbf{u} \times \omega} \right)_{\mathbf{k}}^{t_{n-1}} \right] \Bigg\},\end{aligned} \tag{4.46}$$

$$\begin{aligned}\widehat{\mathbf{u}}_{\mathbf{k}}^{t_{n+1}} = \widehat{\mathbf{u}}_{\mathbf{k}}^{t_n} e^{-\nu k^2 dt} + \frac{dt}{2} \mathbf{P} \Bigg\{ 3 \left[e^{-\nu k^2 dt} \left(\widetilde{\mathbf{u} \times \omega} \right)_{\mathbf{k}}^{t_n} \right] - \\ \left[e^{-\nu k^2 (2dt)} \left(\widetilde{\mathbf{u} \times \omega} \right)_{\mathbf{k}}^{t_{n-1}} \right] \Bigg\},\end{aligned} \tag{4.47}$$

$$\widehat{\mathbf{u}}_{\mathbf{k}}^{t_{n+1}} = e^{-\nu k^2 dt} \left\{ \widehat{\mathbf{u}}_{\mathbf{k}}^{t_n} + \frac{dt}{2} \mathbf{P} \left[3 \left(\widetilde{\mathbf{u} \times \omega} \right)_{\mathbf{k}}^{t_n} - e^{-\nu k^2 dt} \left(\widetilde{\mathbf{u} \times \omega} \right)_{\mathbf{k}}^{t_{n-1}} \right] \right\}. \tag{4.48}$$

Capítulo 5

Cluster

5.1. Introducción

Cluster es un término anglosajón que se refiere a un conjunto de ordenadores conectados entre si creando un único sistema para el cálculo. De este modo se pueden realizar cálculos que de otro modo serían muy lentos o imposibles debido a las limitaciones de velocidad o memoria que puede gestionar un único procesador. Históricamente para el cálculo en paralelo se han diseñado dos tipos de sistemas:

Sistemas tipo supercomputador. Son sistemas diseñados especialmente para el cálculo en paralelo, los equipos disponen de un hardware que interconecta varios procesadores compartiendo el mismo espacio de memoria. Son equipos muy potentes y costosos con un software específico que les permite sacar partido de sus grandes capacidades de procesamiento. Dadas sus características en este tipo de sistemas se usa frecuentemente la programación en hilos. Uno de los principales inconvenientes de este tipo de sistemas es junto con su elevado precio la dificultad de ampliar los recursos ya que son diseñados para un número determinado de procesadores.

Sistemas tipo cluster. Son sistemas construidos a partir de equipos de sobremesa normales, lo cual hace que sean mucho más económicos por lo que se están convirtiendo en una de las opciones más populares. Para la interconexión de los mismos se puede usar una gran variedad de sistemas como por ejemplo una red ethernet. La principal ventaja de estos sistemas es su escalabilidad ya que añadirle nodos no tiene grandes dificultades, pero a cambio este tipo de sistemas son mucho más voluminosos y, debido a los sistemas de comunicación entre nodos, más lentos en la mayor parte de problemas a resolver. La forma más habitual de usar estos sistemas es mediante la programación en paralelo usando un protocolo de paso de mensajes como el MPI.

5.2. Descripción del cluster *Euler*

En este proyecto se va a implementar un cluster formado por 10 nodos, para su interconexión se van a utilizar tarjetas SCI creando un toroide bidimensional (ver figura 5.2) con las conexiones. Las tarjetas SCI de la marca Dolphin Interconnect Solutions Inc. [6] permiten implementar un cluster con características similares a un supercomputador. Sus características principales son su baja latencia (del orden de 1.4 microsegundos) y gran ancho de banda (unos 326 MegaBytes por segundo), lo que nos permite una comunicación eficiente entre los

Figura 5.1: Foto del cluster *Euler*.

procesos de los distintos nodos del cluster. Cada nodo es un ordenador con dos microprocesadores Opteron de la marca AMD y dos gigabytes de memoria con el sistema operativo Debian GNU/Linux [5]. En la figura 5.2 se puede ver la interconexión de los nodos con las tarjetas SCI en la que se puede apreciar que se forman siete anillos, que crean una malla de interconexión de 5x2.

Junto con las tarjetas, se usan las librerías NMPI [15] que implementan el interfaz de paso de mensajes sobre las tarjetas SCI, estas librerías nos ofrecen un sistema sencillo de intercambio de información y sincronización entre los nodos.

El sistema de ficheros del cluster está implementado con NFS, todos los nodos montan el directorio */home* de *Euler-1* que es el primer nodo. De esta forma tenemos un sistema de ficheros único para todos los nodos, además, los usuarios de los distintos nodos están sincronizados mediante un servidor LDAP[†] que asigna un mismo UID y GID así como la misma contraseña a cada usuario. De esta forma en todos los nodos el usuario es el mismo, dando la sensación de una misma máquina. El sistema NFS usa una red ethernet GigaBit. Por tanto, no hay recursos compartidos entre la red NFS y la red MPI.

5.3. Configuración del cluster *Euler*

5.3.1. Sistema operativo

En cada nodo se ha instalado el sistema operativo Debian GNU/Linux [5], se ha instalado la versión *testing* de la distribución para AMD64 y Opteron, actualmente esta distribución no forma parte de la sección oficial de Debian pero se espera que en la próxima versión *etch* sea incluida.

Las características de la instalación del sistema operativo son las siguientes:

- Los nodos se llaman *Euler-1*, *Euler-2* ...
- Sus direcciones IP son 192.168.2.50 en adelante.
- Se instala el sistema mínimo.
- Usan los servidores OPENLDAP Delfos y Sibila para obtener la información de usuarios.

[†]Los servidores OPENLDAP y la configuración de los servidores SSH han sido montados por D. Tomás Manzano Galán, el autor agradece su gran colaboración sin la cual este proyecto habría sido mucho más complicado.

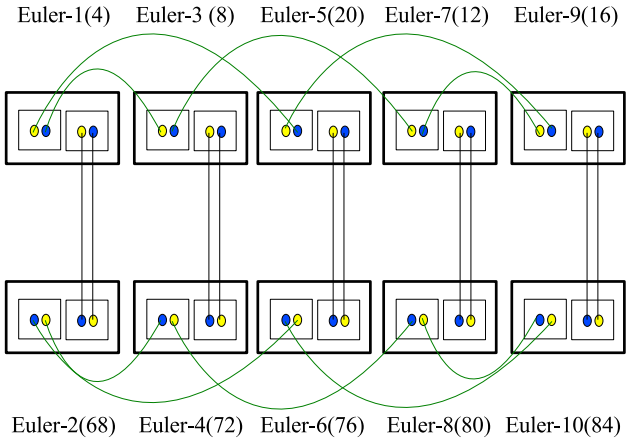

Figura 5.2: Toroide bidimensional de nodos.

- Todos los nodos montan el directorio */home* de *Euler-1* por NFS.
- Se instala el núcleo 2.6.8-11-amd64-k8-smp, y no el paquete virtual que va actualizando el núcleo ya que los drivers de las tarjetas SCI se compilan para un núcleo determinado y su actualización por error podría dejarlos inoperativos.
- Se instalan las fuentes del núcleo.
- Se instala el gcc-3.4. Que es con el que está compilado el núcleo instalado.
- Se instala el paquete gfortran que permite compilar programas en Fortran 95. Si bien este paquete no es necesario para este proyecto, se instala para un posible futuro uso por parte de los usuarios.
- Se instalan adicionalmente los paquetes zlib1g-dev, libmagick9 y libmagick9-dev, necesarios para compilar distintos aspectos del sistema y el programa.

5.3.2. Instalación de las tarjetas SCI

Instalación del hardware

Las tarjetas SCI se instalan en un bus PCI. El cableado asociado, tiene en cuenta que las conexiones se hacen en anillo por lo que, dado el número de nodos, se implementa una configuración de 5x2 que se muestra en la figura 5.2. A la hora de realizar el cableado se ha tenido en cuenta la mínima longitud posible de los cables ya que su coste es elevado.

Compilación e instalación del driver

Para instalar los drivers es necesario descargar sus fuentes del fabricante [6], actualmente el fabricante no soporta la instalación en Debian GNU/Linux, por lo que es necesario hacer algunas modificaciones para su correcta compilación. A continuación se detallan las órdenes mediante las que se ha conseguido compilar el driver partiendo de los paquetes de las fuentes que se deben colocar en el directorio */root/SCI*.

```
export PATH_LINUX_INCLUDE=/usr/src/kernel-headers-2.6.8-11-amd64-k8-smp/include
export PATH_LINUX_CONFIG=/lib/modules/2.6.8-11-amd64-k8-smp/build
cd /root/SCI
tar xzf DIS_RELEASE_3_0_3_OCT_26_2005.tar.gz
cd DIS_RELEASE_3_0_3_OCT_26_2005/src/
tar xzf ../../SCI_SOCKET_3_0_3_OCT_20_2005.tar.gz
```

Con esto hemos descomprimido las fuentes y preparado las variables de entorno necesarias. Antes de poder compilar hay que realizar modificaciones en los siguientes archivos:

- En */SCI/DIS_RELEASE_3_0_3_OCT_26_2005/src/SCI_SOCKET/LINUX/os/headers.h* cambiar *zlib.h* por *linux/zlib.h*.
- En */SCI/DIS_RELEASE_3_0_3_OCT_26_2005/src/SCI_SOCKET/ksocket/lib/Makefile* cambiar *-m32* por *-m64*.
- En */SCI/DIS_RELEASE_3_0_3_OCT_26_2005/src/adm/MAKE/MK-CONFIG-TOOLS-CC-LINUX* cambiar *-m32* por *-m64*.

Una vez hechos estos cambios, seguimos con la compilación

```
cd ../adm/bin/Linux_pkgs
./make_PSB66_X86_64_release
```

Ya está compilado el driver. Ahora lo instalamos y configuramos.

```
cd ../disinst
```

Creamos el fichero */root/Cluster.conf* que tiene el siguiente contenido:

```
Euler-1 4        0       PSB66
Euler-2 68       0       PSB66
Euler-3 8        0       PSB66
Euler-4 72       0       PSB66
Euler-5 12       0       PSB66
Euler-6 76       0       PSB66
Euler-7 16       0       PSB66
Euler-8 80       0       PSB66
Euler-9 20       0       PSB66
Euler-10 84      0       PSB66
```

Y después ejecutamos:

```
./discinst --check </root/Cluster.conf
```

Con lo que comprobamos que el acceso a los nodos esté bien. A continuación instalamos el driver en todos los nodos con el siguiente comando

```
./discinst --install --archive ../Linux_pkgs/DIS_Linux_2.6.8-11-amd64-k8-smp_190106.
   tar.gz < /root/Cluster.conf
```

Instalación manual del driver

A continuación detallamos los pasos a seguir para instalar el driver de forma manual:

```
mkdir /opt/DIS
cp -r DIS_Linux_2.6.8-11-amd64-k8-smp_LATEST/* /opt/DIS
cd /opt/DIS/sbin
./drv-install add PSB66 manager
```

Una vez ejecutados, estos comandos instalan el driver de la tarjeta en el nodo. Este proceso es tedioso y requiere que la carpeta *DIS_Linux_2.6.8-11-amd64-k8-smp_LATEST* que se generó en la compilación esté accesible en cada nodo, por lo que se recomienda seguir los pasos del apartado anterior.

Configuración de las tarjetas SCI

Una vez que hemos instalado el driver, podemos gestionar el cluster mediante una utilidad gráfica llamada SCI Interconnect Manager, esta utilidad que se puede ejecutar en cualquier ordenador (no necesariamente un nodo del cluster) gestiona de forma eficiente la configuración de la topología del cluster.

El programa SCI Interconnect Manager funciona conectándose mediante una red IP a los nodos para poder configurarlos, para su correcto funcionamiento necesita de los programas *SCINodeManager* y *SCINetworkManager*. A continuación se indican los comandos con los que se preparan los nodos para su configuración con SCI Interconnect Manager.

El *SCINodeManager* se ejecuta en todos los nodos, por tanto usamos:

```
./discinst --usercmdbg '/opt/DIS/sbin/scinodemanager -v -sciconfig
   /opt/DIS/sbin/sciconfig -scidiag /opt/DIS/sbin/scidiag -l
   /var/log/scinodemanager.log' < /root/Cluster.conf
```

A continuación es necesario ejecutar el programa *SCINetworkManager* en un nodo, vamos a elegir ejecutarlo en Euler-1:

```
/opt/DIS/sbin/scinetworkmanager -f /etc/dis/SCINetworkManager.conf -dimensionX 5
   -dimensionY 2 -l /var/log/scinetworkmanager.log&
```

Ya tenemos el sistema preparado, ahora podemos usar el SCI Interconnect Manager para analizar el sistema. En la figura 5.3 se puede ver el programa conectado al cluster.

Configuración manual de las tarjetas SCI

Como alternativa a los programas de la sección anterior, se puede configurar el cluster nodo a nodo de forma manual, para lo cual usaremos los programas *SciConfig* y *SciAdmin*.

Con el programa */opt/DIS/sbin/sciconfig* se establece el *NodeId* de cada tarjeta lo cual es imprescindible para su correcto funcionamiento. El *NodeId* de cada tarjeta se calcula de la siguiente tabla que indica el mismo programa:

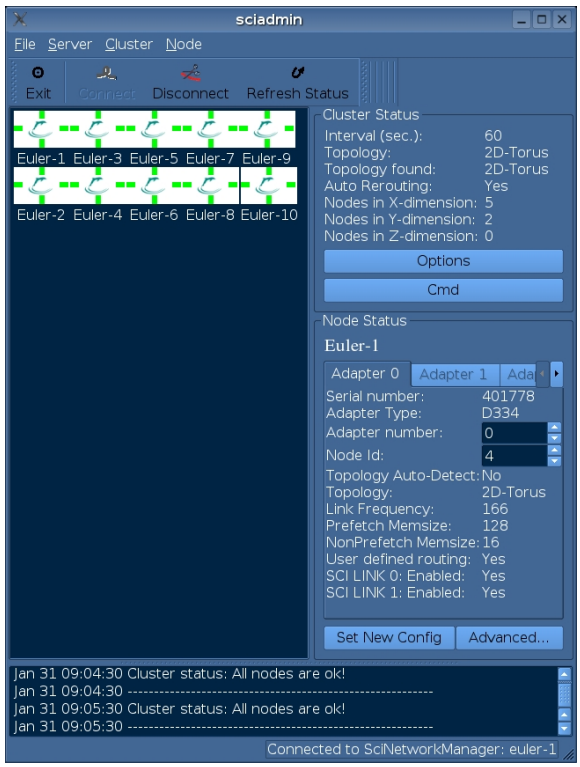

Figura 5.3: Programa SCI Interconnect Manager.

2-D TOPOLOGY

Y	X	NodeIds
0	0-14	4 - 60
1	0-14	68 - 124
2	0-14	132 - 188
3	0-14	196 - 252
4	0-14	260 - 316
5	0-14	324 - 380
6	0-14	388 - 444
7	0-14	452 - 508
8	0-14	516 - 572
9	0-14	580 - 636
10	0-14	644 - 700
11	0-14	708 - 764
12	0-14	772 - 828
13	0-14	836 - 892
14	0-14	900 - 956

Por tanto, usamos los siguientes *NodeId*:

Nodo	Euler-1	Euler-2	Euler-3	Euler-4	Euler-5
NodeId	4	68	8	72	12
Nodo	Euler-6	Euler-7	Euler-8	Euler-9	Euler-10
NodeId	76	16	80	20	84

Para poder comprobar el estado de un nodo podemos usar el programa */opt/DIS/sbin/scidiag* que nos dará un informe del estado del nodo como el siguiente.

```
Euler-1:/opt/DIS/sbin# ./scidiag
============================================================================
        SCI diagnostic tool --  SciDiag version 3.1.1 ( December 20th 2005 )
============================================================================

******************** VARIOUS INFORMATION              ********************
Driver: Dolphin IRM 3.1.1 (  December 20th 2005 )
Scidiag compiled in 64 bit mode
Date  : mar ene 31 09:22:29 CET 2006
System: Linux Euler-1 2.6.8-11-amd64-k8-smp #1 SMP Sun Oct 2 23:21:12 CEST 2005
   x86_64 GNU/Linux

Number of configured local adapters found: 1

Hostbridge : AMD-8131 , 0x74501022
Local adapter 0 > Type                : D334
                  NodeId(log)         : 4
                  NodeId(phys)        : 0x4
                  SerialNum           : 401778
                  PSB Version         : 0x0d66706d
                  LC Version          : 0x1066606d
                  PLD Firmware        : 0x0000
                  IO Bus frequency    : 66 MHz
                  SCI Link frequency  : 166 MHz
                  B-Link frequency    : 80 MHz
                  Card Revision       : CD
                  Switch Type         : not present
                  Topology Type       : 2D Torus
                  Topology Autodetect : No
OK: Psb chip alive in adapter 0.
SCI Link 0 - uptime 762 seconds
SCI Link 1 - uptime 762 seconds
OK: Cable insertion ok.
ioctl failed : IOC_GET_LC_GEO_REG (GetLocalLcCsr): No error
Problem: SCI Link 2: Undefined Lc.InitSt.initstate: 0xd
OK: LC error count has constant value.
OK: Probe of local node ok.
OK: Link alive in adapter 0.
OK: SRAM test ok for Adapter 0
OK: LC-3 chip accessible from blink in adapter 0.
==> Local adapter 0 NOT ok!
```

```
******************** TOPOLOGY SEEN FROM ADAPTER 0     ********************
Adapters found: 10   Switch ports found: 0
----- List of all adapters and switches found:
Sci adapter>  NodeId: 0004  Scrubber: 0  BlinkId: 0          <----- On Local Host
Sci adapter>  NodeId: 0008  Scrubber: 0  BlinkId: 0
Sci adapter>  NodeId: 0012  Scrubber: 0  BlinkId: 0
Sci adapter>  NodeId: 0016  Scrubber: 0  BlinkId: 0
Sci adapter>  NodeId: 0020  Scrubber: 0  BlinkId: 0
Sci adapter>  NodeId: 0068  Scrubber: 0  BlinkId: 0
Sci adapter>  NodeId: 0072  Scrubber: 0  BlinkId: 0
Sci adapter>  NodeId: 0076  Scrubber: 0  BlinkId: 0
Sci adapter>  NodeId: 0080  Scrubber: 0  BlinkId: 0
Sci adapter>  NodeId: 0084  Scrubber: 0  BlinkId: 0
----- List of all ranges (rings) found:
In range 0:  0004  0008  0012  0016  0020
In range 1:  0068  0072  0076  0080  0084
---------------------------------
scidiag discovered 0 note(s).
scidiag discovered 0 warning(s).
scidiag discovered 1 error(s).
TEST RESULT: *FAILED*
```

En este informe correspondiente al nodo Euler-1, podemos comprobar el estado de los cables y la información de configuración. En este caso se ha detectado un error en el Link 2, este error no afecta ya que las tarjetas de las que dispone el equipo tienen sólo Links 0 y 1, se puede resolver este error desactivando manualmente el Link 2 en el archivo de configuración */etc/dis/SCINetworkManager.conf.*

Enlace de las librerías dinámicas

Para poder compilar el programa NMPI, así como cualquier programa que intente usar las librerías dinámicas que se compilan con el driver, es necesario indicarle al sistema dónde están los archivos, esto se hace en el archivo */etc/ld.so.conf* en el que hay que añadir la siguiente línea:

```
/opt/DIS/lib
```

Después hay que ejecutar el comando:

```
ldconfig -v
```

Configuración e instalación de NMPI

El paquete NMPI es una implementación del protocolo MPI que nos permite utilizar las tarjetas SCI, este paquete es una serie de modificaciones al paquete MPICH2 [14] que hacen que las comunicaciones sean a través del interfaz SCI. Para instalar el paquete NMPI, obtenemos las fuentes de la web del fabricante [15] y ejecutamos los siguientes comandos:

```
tar xzf nmpi-1.2.tar.gz
cd nmpi-1.2
export CFLAGS="-fPIC"
export C90FLAGS="-fPIC"
export CXXFLAGS="-fPIC"
./configure -with-device=ch3:stream --with-sisci=/opt/DIS --prefix /usr
   --enable-f90 --enable-sharedlibs=gcc --enable-romio --enable-runtimevalues
```

Las fuentes que hemos descargado tienen un problema de configuración y no compilan bien las librerías compartidas. Como en este proyecto nos interesa usarlas con el paquete fftw, hay que modificar en este punto los ficheros *nmpi-1.2/src/mpid/ch3/channels/stream/src/Makefile* y *nmpi-1.2/src/mpid/ch3/channels/stream/streams/Makefile* para cambiar la línea en la que se define la variable `CXX_SHL` por la siguiente:

```
CXX_SHL= c++ -shared -fPIC
```

Además en el archivo *nmpi-1.2/src/mpid/ch3/channels/stream/streams/streamsconfig.hxx* hay que quitar los comentarios de la línea

```
#define _REENTRANT
```

Una vez realizados los cambios el programa compila finalmente y con las siguientes órdenes se termina la instalación:

```
make
make install
```

En el caso de que se desee desinstalar el programa NMPI hay que usar la orden:

```
/usr/sbin/mpeuninstall
```

Como en el caso del driver hay que ejecutar la orden

```
ldconfig
```

A continuación hay que proceder a la configuración del software tal y como se hace con la distribución de MPICH2. Se crea el fichero *.mpd.conf* en el directorio raíz del usuario o el fichero */etc/mpd.conf* para el usuario root. En este fichero se pone una palabra clave cualquiera, por ejemplo se usa:

```
secretword=Prueba
```

Después nos aseguramos de que el fichero pueda ser leído con el siguiente comando:

```
chmod 600 .mpd.conf
```

o para root:

```
chmod 600 /etc/mpd.conf
```

Para que se puedan ejecutar los demonios mpd en cada nodo hay que crear el archivo *mpd.host* en el directorio raíz:

```
Euler-1
Euler-2
Euler-3
Euler-4
Euler-5
Euler-6
Euler-7
Euler-8
Euler-9
Euler-10
```

Antes de comprobar si funciona el cluster, es interesante comprobar si el NMPI funciona en un nodo, para lo cual tecleamos:

```
mpd &
mpdtrace
mpdallexit
```

Una vez comprobamos que no hay errores, continuamos comprobando que se puedan realizar llamadas SSH entre los nodos sin autentificación de usuario, para lo cual se configuran los servidores SSH de los nodos adecuadamente. También se podrían haber usado en vez de servidores SSH servidores RSH o TELNET, pero se ha optado por motivos de seguridad por el protocolo SSH ya que no supone un gasto apreciable en rendimiento y ofrece unas comunicaciones de red mucho más seguras. Por último, detallamos los pasos que han de seguir los usuarios para utilizar el entorno MPI:

1. Ejecutar los servidores MPD en cada nodo, para eso en cualquier nodo se ejecuta la orden *mpdboot -n <Nº de nodos>*.

2. Comprobar que los servidores se han activado adecuadamente con la orden *mpdtrace*.

3. Ejecutar el programa con la siguiente llamada: *mpirun -n <Nº de procesos> <Ruta del programa>*.

4. Cerrar los servidores MPD de cada nodo ejecutando en cualquier nodo la orden *mpdallexit*.

5.3.3. Instalación de fftw

El paquete fftw que viene en la distribución de Debian no se puede instalar porque depende de las librerías MPICH que no se han instalado ya que se usa el paquete NMPI que no pertenece a la distribución de Debian GNU/Linux. Por tanto hay que compilarlo e instalarlo manualmente para que funcione con el NMPI, para lo cual seguimos los siguientes pasos:

```
mkdir fftw
cd fftw
apt-get source fftw2
cd fftw-2.1.3/
./debian/rules binary
cd ..
dpkg -i *.deb
```

Capítulo 6

Implementación del código numérico

En este capítulo se va a describir las características de la simulación y de la implementación del código así como los puntos donde se puede aplicar la paralelización de los cálculos.

6.1. Introducción

El programa divide el cubo en una serie de puntos equiespaciados en forma de mayado ortogonal en los que se realizan los cálculos. En cada punto del cubo se define el vector de velocidades con sus tres componentes y la parte principal del programa se encarga de controlar la evolución temporal del campo de velocidades en el cubo. Para realizar esta evolución, recurrimos a modelar la dinámica del sistema con las ecuaciones de Navier-Stokes (ecuación 4.1) junto a la imposición de que el cubo no tiene ni fuentes ni sumideros, o lo que es lo mismo, que su divergencia es cero (ecuación 4.2). En el capítulo 4 se detallan las expresiones que conducen a la ecuación que dado un estado nos permite obtener el siguiente (ecuaciones 4.42 y 4.48). El objetivo de ese capítulo es encontrar un método que sea eficiente computacionalmente hablando.

En una transformada DFT como la que usamos en este proyecto, obtenemos términos en frecuencia que se corresponden con valores de $|\mathbf{k}|$ que van desde 0 en el centro hasta $\sqrt{\frac{3}{4}}N$ en las esquinas del cubo. Debido a los errores numéricos los términos correspondientes a valores de $|\mathbf{k}|$ altos tienen una gran componente de ruido por los fenómenos de aliasing y truncamiento de la serie, y no son de interés para esta simulación. Por esto, al igual que en el artículo de L. P. Wang y M. R. Maxey [23], se aplica un filtro esférico que elimina las componentes en frecuencia del campo de velocidades en las que $|\mathbf{k}| > \frac{N}{2} - 1{,}5$.

El forzado en esta simulación se va a realizar introduciendo el concepto de viscosidad negativa en las componentes espectrales de módulo menor que $2,25$; es decir, en los cálculos que se realizan para obtener el siguiente estado, sustituimos la viscosidad de los términos en los que $|\mathbf{k}| < 2{,}5$ por cierto valor negativo. Este valor se calcula mediante un PID [16] de forma que se mantenga un estado estacionario en el que la energía que se aporta al sistema sea igual a la que se disipa. Como variable de control del PID se elige que el producto $K_{max}\eta \simeq \frac{N}{2}\eta$ sea un valor constante e igual a 1,4 (aunque este valor es configurable).

De los artículos publicados sobre esta materia, se sabe que para estas simulaciones el paso de tiempo óptimo es $dt = 0{,}025T_\eta = 0{,}025\left(\frac{\nu}{\epsilon}\right)^{\frac{1}{2}}$, por eso el código mantiene el paso en el intervalo $0{,}015 < \frac{dt}{T_\eta} < 0{,}025$.

6.2. Codificación

El código se ha programado en lenguaje C [13] que junto con el Fortran son los lenguajes más comunes en la programación de simulaciones numéricas. En la implementación de las distintas características se van a usar tres librerías que nos facilita la codificación.

Librería FFTW

Como comentamos en la sección 3.2.5, esta librería implementa la transformada FFT. Utilizamos esta librería por ser una de las implementaciones del algoritmo más eficientes. Esta eficiencia se basa en el uso de distintas subrutinas específicas para el procesador en el que se compila así como en un análisis previo de las mejores opciones en cuanto al tamaño de la segmentación y otros parámetros internos para obtener la máxima velocidad posible.

Librería NMPI

Como comentamos en la sección 5.3 esta librería implementa el interfaz de paso de mensajes MPI, está basada en el paquete MPICH2 con ciertas modificaciones para usar las tarjetas SCI. Esta librería es con la que llevamos a cabo la paralelización del código, no sólo por las funciones que nos permiten sincronizar la ejecución en los nodos y el intercambio de información, sino porque también se integran con la librería FFTW descargando de la gran complejidad que supone hacer un código optimizado para calcular una FFT en paralelo.

Librería ImageMagick

Si bien esta librería no es necesaria para la simulación, sí nos permite obtener resultados gráficos de la misma. Esta librería se usa para generar las imágenes del espectro, los invariantes y la evolución de $\frac{N}{2}\eta$ en las rutinas de resultados. La ventaja de realizar las gráficas con el programa son que además de dar información en tiempo real del estado de la simulación, en el caso de los invariantes, nos permite calcular la gráfica utilizando el cluster lo cual debido al enorme tamaño del archivo de invariantes es casi necesario.

6.3. Diagramas de flujo

El programa principal tiene el diagrama de flujo de la figura 6.1. Como se puede ver, hay tres bloques:

- El primero es en el que se preparan las condiciones iniciales. Se inicializa el entorno MPI, se reserva la memoria suficiente para definir el estado, se inicializa, y se da el primer paso con el método de Runge-Kutta de segundo orden.
- En el segundo es en el que realizamos la evolución temporal así como la extracción de datos y parámetros en estudio.
- En el tercero se guarda el estado para un futuro uso, y se liberan los recursos del cluster.

En la figura 6.2 se puede ver en detalle el diagrama de flujo del bloque principal. A continuación vamos a describir las características más destacadas de la implementación.

El algoritmo de Adams-Bashforth descrito por la ecuación 4.48 al ser de segundo orden, necesita de las variables en dos instantes para calcular el siguiente, por tanto la memoria que necesitamos reservar es:

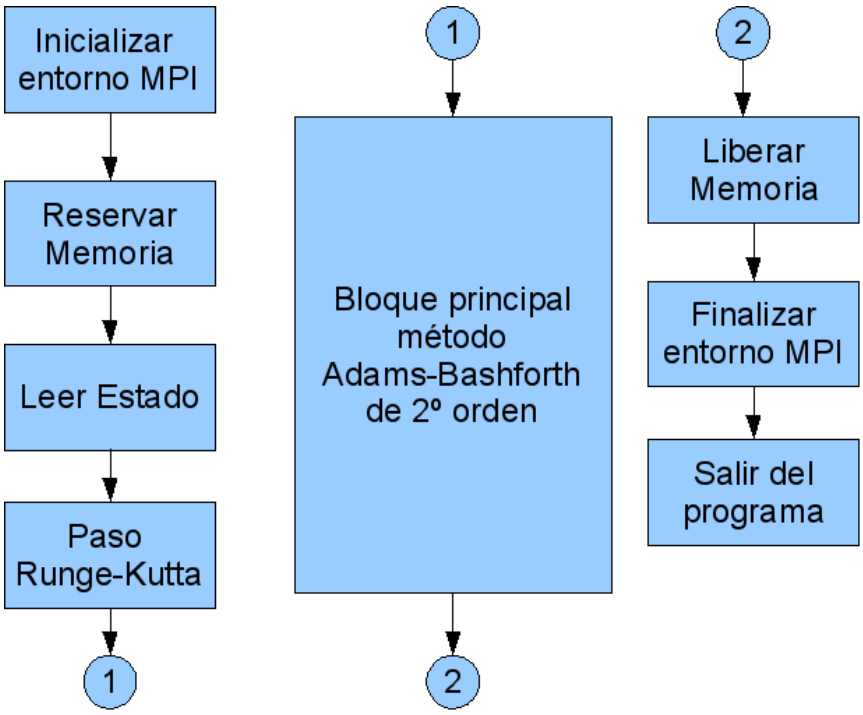

Figura 6.1: Diagrama de flujo.

- Estado actual:
  - Vector de velocidades: $\widehat{\mathbf{v}}_k^{t_n}$ Memoria: $3 * N^3$
  - Término no lineal: $\left(\widehat{\mathbf{v} \times \omega}\right)_k^{t_n}$ Memoria: $3 * N^3$
  - Espectro: Memoria: $N/2$
  - Matriz de invariantes: Memoria: $9 * N^3$
  - Cálculo de imágenes: $640 * 480$ de fondos + 1000 de gráfica de evolución del $K\eta$
- Estado anterior: Memoria: $3 * N^3$
  - Término disipativo: $\left(\widehat{\mathbf{v} \times \omega}\right)_k^{t_{n-1}}$ Memoria: $3 * N^3$
- Nuevo estado:
  - Vector de velocidades: $\widehat{\mathbf{v}}_k^{t_{n+1}}$ Memoria: $3 * N^3$. Este vector también se usa como variable temporal durante el cálculo para ahorrar memoria.

Es decir, el orden de las necesidades de memoria es $21 * N^3$ números reales (en precisión simple para $N = 64$ son aproximadamente 22 MBytes, para $N = 128$ son aproximadamente 176 MBytes,...). La memoria necesaria para las matrices de tamaño N^3 la dividimos entre los nodos del cluster, para ello, usamos la función *rfftwnd_mpi_local_sizes* de la librería fftw que nos permite obtener los valores de offset y tamaño de la matriz local a cada nodo.

Por motivos de eficiencia computacional, la librería fftw la vamos a configurar de forma que la transformada de Fourier calcula la matriz transpuesta de la que queremos que calcular,

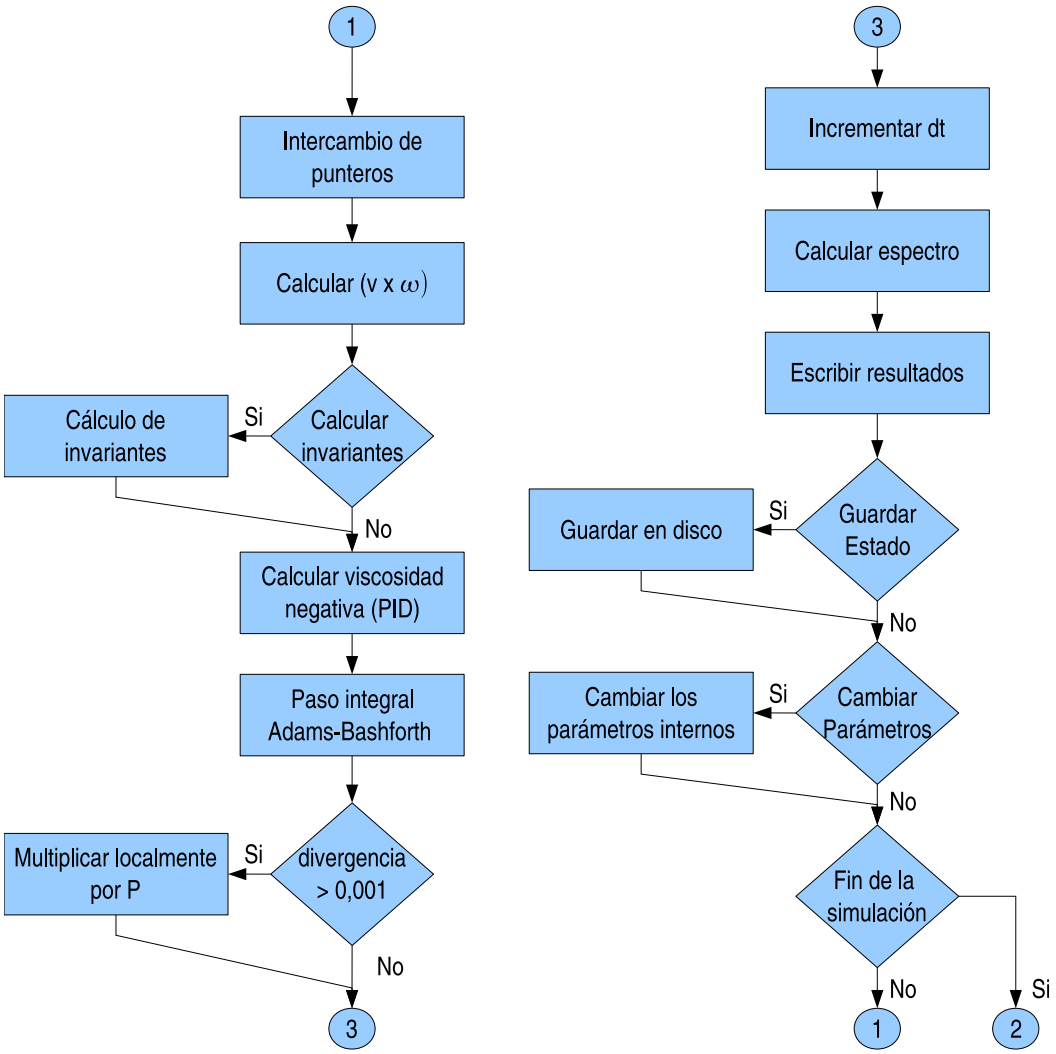

Figura 6.2: Detalle del diagrama de flujo.

mediante esta pequeña complicación en el código, ahorramos una gran cantidad de intercambios de bloques de memoria entre los nodos. Se puede ver un pequeño ejemplo del uso de la librería fftw en el cuadro de código 6.1.

Los pasos a seguir para realizar una transformada con la librería fftw son

1. Crear los planes de ejecución con la función *rfftw3d_mpi_create_plan*.

2. Calcular el tamaño de la memoria necesaria en cada nodo y los valores de offset respecto a la original con la función *rfftwnd_mpi_local_sizes*.

3. Reservar la memoria necesaria, para ahorrar memoria, en las transformadas de valores reales, la librería utiliza la mitad de la memoria, pero si el número de muestras es par se usa un poco más de memoria, dada una secuencia de tamaño N, la cantidad de memoria que se necesita es [2*(N/2+1)].

4. Inicializar los datos.

5. Transformar con la función *rfftwnd_mpi*.

6. Usar los datos.

7. Liberar la memoria.

8. Destruir los planes con *rfftwnd_mpi_destroy_plan*.

```
// Inicializamos el entorno MPI
MPI_Init(&argc,&argv);
//Creamos los planes
Plan_mpi = rfftw3d_mpi_create_plan(MPI_COMM_WORLD, N, N, N,
    FFTW_REAL_TO_COMPLEX, FFTW_ESTIMATE);
PlanInv_mpi = rfftw3d_mpi_create_plan(MPI_COMM_WORLD, N, N, N,
    FFTW_COMPLEX_TO_REAL, FFTW_ESTIMATE);
//Obtenemos los tamaños de las matrices en el nodo
rfftwnd_mpi_local_sizes(Plan_mpi, &TAM_LOCAL.NX, &TAM_LOCAL.
    NX_offset, &TAM_LOCAL.NY, &TAM_LOCAL.NY_offset, &TAM_LOCAL.
    TOTAL);
//Reservamos memoria
VX.Esp = fftw_malloc(sizeof(fftw_real) * TAM_LOCAL.TOTAL);
//Espacio de memoria auxiliar para aumentar la velocidad
EspacioAux.Esp = fftw_malloc(sizeof(fftw_real) * TAM_LOCAL.TOTAL
    );
//Inicialización de los datos (campo aleatorio)
srand(MPI.PROCESO_MPI);
for(x=0;x<TAM_LOCAL.NX;x++)
    for(y=0;y<N;y++)
        for(z=0;z<N;z++)
        {
            if(rand()>RAND_MAX/2.0)
                VX.Esp[z+(2*((N/2)+1))*(y+N*x)]=AMPLITUD_INICIAL*(
                    (1.0*rand())/(RAND_MAX+1.0));
            else
                VX.Esp[z+(2*((N/2)+1))*(y+N*x)]=-AMPLITUD_INICIAL*(
                    (1.0*rand())/(RAND_MAX+1.0));
        }
//Ejemplo de transformada directa
rfftwnd_mpi(Plan_mpi, 1, VX.Esp, EspacioAux.Esp,
    FFTW_TRANSPOSED_ORDER);
//Ejemplo de transformada inversa
rfftwnd_mpi(PlanInv_mpi, 1, VX.Esp, EspacioAux.Esp,
    FFTW_TRANSPOSED_ORDER);
//Liberarmos la memoria
free(VX.Esp);
free(EspacioAux.Esp);
rfftwnd_mpi_destroy_plan(Plan);
rfftwnd_mpi_destroy_plan(PlanInv);
//Terminamos el entorno MPI
MPI_Finalize();
```

Código fuente 6.1: Ejemplo de transformada con fftw.

El programa utiliza dos archivos de parámetros para gestionar el cambio en tiempo real de los parámetros de la simulación (ver apartado 6.4) como la viscosidad, el tamaño del filtro, etc.

Dado la gran cantidad de datos numéricos que son los resultados de este programa, se ha optado porque además de los archivos con los valores, el programa genere un resultado gráfico que ayude a analizar los mismos. Para crear las imágenes se utiliza la librería ImageMagick que nos permite realizar distintos tipos de gráficas. Un breve tutorial de su uso sería el siguiente:

1. Reservar un espacio en memoria del tamaño de la imagen.
2. Crear el objeto MagickWand que es el que genera la imagen.
3. Crear el objeto PixelWand que nos permite seleccionar las características de estilo de los objetos que se dibujan.
4. Crear el objeto DrawingWand que hace de interfaz de dibujo en el objeto MagickWand usando las características de PixelWand.
5. Usar las funciones de dibujo que permiten modificar las características del PixelWand, por ejemplo *PixelSetColor*.
6. Usar las funciones de dibujo que permiten pintar en el dibujo, por ejemplo *DrawLine*.
7. Crear la imagen en disco con *MagickWriteImages*.
8. Destruir los objetos DrawingWand, PixelWand y MagickWand.

6.4. Configuración en tiempo real

Como se ha comentado, el programa admite el ajuste de sus parámetros internos en tiempo real, para lo cual se utilizan dos archivos

- Archivo de entrada de parámetros: *Parametros.dat*. En este archivo se le pasan las indicaciones al programa para que modifique alguno de los parámetros internos.
- Archivo de indicación de parámetros: *Parametros-salida.dat*. En este archivo el programa va indicando los valores de los parámetros en cada paso de integración.

El formato de los parámetros es el mismo en ambos archivos, son archivos de texto en los que cada línea indica un parámetro, se van leyendo consecutivamente por lo que si se repite alguno el que tiene efecto es el último. Los parámetros que se pueden pasar son:

- FORZADO: Indica si se usa el forzado para inyectar energía al sistema o se deja la evolución libre.
- CALCULAINVARIANTES: Indica si se deben calcular los invariantes.
- GUARDAESTADO: Indica si se debe guardar el estado en disco.
- Pasos: Indica cuántos pasos integrales debe dar el programa.
- VISCOSIDAD: Valor de la viscosidad.

- dt: Valor del siguiente paso de integración. Si se usa este parámetro, el siguiente paso integral se da usando el método de Runge-Kutta con el nuevo valor.
- K_FILTRO: Valor del $|k|$ de los mayores modos que se calculan. Los modos que tengan un $|k|$ mayor se fuerza a que valgan cero.
- CURSOR_K: Indica dónde se pone el cursor vertical que sirve para medir sobre la gráfica del espectro.
- CURSOR_E: Indica dónde se pone el cursor horizontal que sirve para medir sobre la gráfica del espectro.
- KETA_ESTACIONARIA: Indica el valor de $k_{max}\eta$ al que el PID intenta estabilizar el sistema.
- TIEMPOTOTAL: Indica el tiempo total transcurrido de integración.
- PID_K_P: Valor de la constante proporcional del PID.
- PID_K_I: Valor de la constante integral del PID.
- PID_K_D: Valor de la constante derivativa del PID.
- PID_error: Valor del error en el paso de integración que se usa junto con la constante proporcional en PID.
- PID_deriv_error: Valor de la derivada del error en el paso de integración que se usa junto con la constante derivativa en PID.
- PID_int_error: Valor de la integral del error en el paso de integración que se usa junto con la constante integral en PID.
- Imagen_Ancho: Ancho de las imágenes que se generan.
- Imagen_Alto: Alto de las imágenes que se generan.
- Imagen_KETA_MAX: Valor de $k_{max}\eta$ máximo que aparece en la gráfica de $k_{max}\eta$.
- Imagen_KETA_MIN: Valor de $k_{max}\eta$ mínimo que aparece en la gráfica de $k_{max}\eta$.
- Escala_Invariantes_X: Factor por el que se multiplican los valores del determinante en la gráfica de los invariantes.
- Escala_Invariantes_Y: Factor por el que se multiplican los valores del invariante asociado a los adjuntos en la gráfica de los invariantes.
- Imagen_Divisiones_X: Número de décadas en la gráfica del espectro.
- Imagen_Divisiones_Y: Número de divisiones en el eje Y en la gráfica del espectro.

Para poder gestionar todos estos parámetros se ha realizado un pequeño programa que lee los parámetros de salida y nos permite ajustar los parámetros y generar el archivo de paso de parámetros cómodamente. En la figura 6.3 se puede ver el funcionamiento del programa. El programa llamado *ajustanavier* ha sido diseñado para el entorno KDE bajo entornos Linux. Su uso es muy simple, se debe ejecutar en el mismo directorio que el programa principal y se sincroniza automáticamente. En realidad, el programa *ajustanavier* se suele ejecutar en un ordenador que no pertenece al cluster ya que este no dispone de entorno gráfico, lo que se hace es montar el directorio de trabajo del cluster en el ordenador del usuario mediante la red NFS y ejecutar el programa de configuración en el entorno gráfico del mismo.

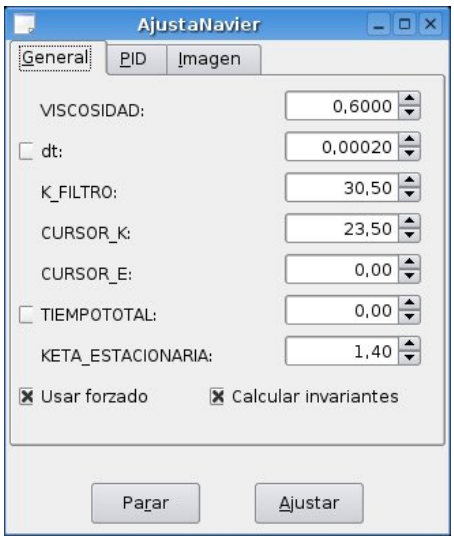

Figura 6.3: Programa de ajuste de parámetros internos en tiempo real.

6.5. Compilación y ejecución del programa

El código está estructurado en distintos archivos fuente, para compilar y crear el ejecutable a partir de los mismos ejecutamos una de las siguientes órdenes, la diferencia entre ambas es que la primera genera un código con precisión doble y el segundo con precisión simple.

```
mpicc -o navier_mpi_doble -lm -lfftw -lrfftw -lMagick -lWand
   -lrfftw_mpi -lfftw_mpi *.c
```

Código fuente 6.2: Generación del ejecutable con precisión doble a partir del código fuente.

```
mpicc -o navier_mpi_simple -DPrecisionSimple -lm -lsfftw -
    lsrfftw
   -lMagick -lWand  -lsrfftw_mpi -lsfftw_mpi -lfftw -lfftw_mpi -
       lrfftw
   -lrfftw_mpi *.c
```

Código fuente 6.3: Generación del ejecutable con precisión simple a partir del código fuente.

6.6. Paralelización

6.6.1. Introducción

El programa que vamos a implementar tiene bastantes cálculos que se pueden realizar de forma paralela, se puede dividir la forma en la que se va a abordar la paralelización en tres tipos de casos.

Intercambio de parámetros: En este tipo de casos, como puede ser el paso de los valores para iniciar una nueva simulación (N, viscosidad, etc.), se usan las funciones adecuadas del interfaz MPI.

Cálculos en cada punto: En este tipo de casos, como pueden ser los pasos integrales, los cálculos únicamente necesitan de las propiedades de cada punto del cubo, en este caso cada nodo opera sobre los puntos del cubo que tiene en su memoria local. No es necesario ningún tipo de sincronización entre los nodos.

Cálculos de suma o media en todos los puntos del cubo: En este tipo de casos, como puede ser el cálculo del espectro, cada nodo realiza las operaciones necesarias en los puntos de su memoria local y al final se sincroniza con los demás para realizar la suma o media total.

Cálculos en los que están involucrados todos los puntos del cubo: En este tipo de casos la paralelización resulta muy complicada, en nuestro código únicamente se presenta este tipo de casos en los cálculos de los coeficientes de las series de Fourier, y de su correcta y eficiente resolución se ocupa la librería fftw.

6.6.2. Uso de índices

Como ya hemos comentado, las librerías fftw se encargan del cálculo de la FFT usando MPI internamente. Por motivos de eficiencia computacional, las matrices tridimensionales en frecuencia se calculan traspuestas, por tanto, teniendo en cuenta que los bucles están traspuestos y el distinto tamaño de las matrices en los dos dominios, los bucles para recorrer las matrices son:

En el espacio físico

Las variables x, y, z son los índices que representan la posición del punto en el cubo.

```
for(x=0;x<TAM_LOCAL.NX;x++)
{
    for(y=0;y<N;y++)
    {
        for(z=0;z<N;z++)
        {
            indice_Matriz=z+(2*((N/2)+1))*(y+N*x);
            .
            .
            .
```

Código fuente 6.4: Ejemplo de recorrido de matrices en el espacio físico.

En el espacio de Fourier

Las variables x, y, z sirven para desplazarse por el cubo en memoria, y las variables indice_x, indice_y e indice_z se corresponden con k_x, k_y y k_z.

```
for(y=0;y<TAM_LOCAL.NY;y++)
{
    indice_y=((y+TAM_LOCAL.NY_offset)>=(N/2))?((y+TAM_LOCAL.
        NY_offset)-N):(y+TAM_LOCAL.NY_offset);
    for(x=0;x<N;x++)
    {
        indice_x=(x>=(N/2))?(x-N):x;
        for(z=0;z<((N/2)+1);z++)
        {
            indice_z=(z>=(N/2))?(z-N):z;
            indice_Matriz=z+((N/2)+1)*(x+N*y);

            modulo_cuadrado_k=indice_z*indice_z+indice_x*indice_x+
                indice_y*indice_y;
            .
            .
            .
```

Código fuente 6.5: Ejemplo de recorrido de matrices en el espacio de Fourier.

6.6.3. Uso de la librería MPI

La librería se utiliza para la sincronización de los cálculos entre los distintos nodos. En esta sección se va a describir su uso en el código. Lo primero que hay que hacer es la inicialización de entorno MPI. Para ello se utilizan unas funciones (ver código fuente 6.6) que además de inicializar el entorno guardan la información referente a los procesos en una estructura que luego se utiliza para ajustar las matrices locales de cada nodo.

```
if(MPI_Init(&argc, &argv)!=MPI_SUCCESS)
{
    printf("Error: No se puede inicializar el MPI\n"); fflush(
        stdout);
    exit(1);
}
if(MPI_Comm_size(MPI_COMM_WORLD, &MPI.NUMERO_DE_PROCESOS_MPI)!=
    MPI_SUCCESS)
{
    printf("Error: No se puede inicializar el MPI\n"); fflush(
        stdout);
    exit(1);
}
if(MPI_Comm_rank(MPI_COMM_WORLD, &MPI.PROCESO_MPI)!=MPI_SUCCESS)
{
    printf("Error: No se puede inicializar el MPI\n"); fflush(
        stdout);
    exit(1);
}
```

Código fuente 6.6: Ejemplo de inicialización del entorno MPI.

Cuando se desee finalizar el programa, hay que cerrar el entorno MPI con la función:

```
MPI_Finalize();
```

Código fuente 6.7: Ejemplo de cierre del entorno MPI.

Para sincronizar los nodos se puede utilizar la función *MPI_Barrier* que hace que todos los nodos esperen en ese punto del código a que los otros lleguen. Esta función se utiliza al principio de cada bloque de operaciones para que todos los nodos empiecen a la vez el bloque.

```
MPI_Barrier(MPI_COMM_WORLD);
```

Código fuente 6.8: Ejemplo de sincronización de nodos.

Existen situaciones en las que el nodo principal, que en nuestro programa es el número cero, debe comunicar algún parámetro al resto, para ello se utiliza la función *MPI_Bcast*, en el siguiente ejemplo el proceso cero lee el valor de la variable N de la entrada estándar y luego la difunde al resto.

```
if(MPI.PROCESO_MPI==0)
{
    printf("\nValor de N: ");
    z=scanf("%d",&N);
    if(z==0)
    {
        printf("\nValor incorrecto");
        exit(1);
    }
}
MPI_Barrier(MPI_COMM_WORLD);
MPI_Bcast ( &N, 1, MPI_INTEGER, 0, MPI_COMM_WORLD);
```

Código fuente 6.9: Ejemplo de difusión de datos a los nodos.

En los casos en los que hay que realizar una serie de cálculos sobre todos los puntos del cubo, lo que se hace es que cada nodo realiza los suyos sobre una zona de memoria temporal, y luego utiliza la función *MPI_Allreduce* en la que se indica el tipo de operación que se realiza sobre los datos. En el ejemplo de código fuente 6.10 se ve la llamada que se usa para calcular el espectro del sistema (ver sección 7.4); en la zona de memoria apuntada por *Espectro.Valores_Rodaja_MPI* cada nodo ha realizado los cálculos asociados a sus puntos, y lo que hace este código es sumar en cada elemento de la zona de memoria apuntada por *Espectro.Valores_Rodaja* los valores equivalentes en las zonas *Espectro.Valores_Rodaja_MPI* de todos los nodos.

```
MPI_Barrier(MPI_COMM_WORLD);
MPI_Allreduce ( Espectro.Valores_Rodaja_MPI,Espectro.
    Valores_Rodaja, Espectro.tam, MPI_DOUBLE,MPI_SUM,
    MPI_COMM_WORLD);
```

Código fuente 6.10: Ejemplo de sincronización de datos entre los nodos.

Por último, queda comentar la forma en la que se usan los datos en disco, para ello se utilizan funciones de manejo de ficheros que ofrece el interfaz MPI. En el código fuente 6.11 se puede ver un ejemplo de acceso al disco para leer un parámetro.

```
//Variable para manejar ficheros con MPI:
MPI_File DATOS_MPI;
//Sincronizamos los nodos
MPI_Barrier(MPI_COMM_WORLD);
//Abrimos el fichero
if(MPI_File_open (MPI_COMM_WORLD, Nombre, MPI_MODE_RDONLY,
   MPI_INFO_NULL, &DATOS_MPI)!=MPI_SUCCESS)
{
   if(MPI.PROCESO_MPI==0)
   {
      fprintf(stderr, "\nNo puedo abrir %s\n",Nombre);
   }
   exit(1);
}
//Leemos un parámetro
MPI_File_read_all(DATOS_MPI, &N,sizeof(unsigned int),
   MPI_CHARACTER, &MPI.Estado);
//Cerramos el fichero
MPI_File_close(&DATOS_MPI);
```

Código fuente 6.11: Ejemplo de manejo de ficheros con MPI.

Capítulo 7

Resultados

En este capítulo vamos a describir los resultados de los cálculos que se realizan en el programa.

7.1. Estado

El programa principal gestiona la evolución temporal del campo de velocidades en un cubo, el algoritmo se puede ver en la figura 6.2 y en las ecuaciones 4.42 y 4.48 se indica los procedimientos seguidos para integrar. Para poder hacer otro tipo de cálculos o para poder continuar la simulación en otro momento, se guarda un archivo con los datos necesarios para describir el estado del sistema. Este archivo se genera en el directorio desde el que se ejecuta el programa y tiene como nombre *MPI.Estado-NXX.dat* donde *XX* hay que sustituirlo por el valor de N de la simulación. En el cuadro 7.1 se puede ver la estructura del archivo:

Variable	**Tipo**	**Descripción**
N	Entero	Número de divisiones del cubo.
N3D	Entero	Tamaño de las matrices tridimensionales.
N3D_DIV	Entero	Factor de escala para normalizar las transformadas.
N3D_FREC	Entero	Tamaño de las matrices en frecuencia.
K_FILTRO	Real	Valor de k máximo que deja pasar el filtro esférico.
Kcuadrado_FILTRO	Real	Cuadrado del valor anterior.
K_VISCOSIDAD	Real	Valor de k por debajo del cual se introduce energía.
K_VISCOSIDAD_CUAD	Real	Cuadrado del valor anterior.
VISCOSIDAD	Real	Valor de ν.
VISCOSIDADNEG	Real	Último valor calculado de la viscosidad negativa.
EPSILON	Real	Último valor calculado de ϵ.
ETA	Real	Último valor calculado de η.
KETA_ESTACIONARIA	Real	$k_{max}\eta$ al que el PID intenta estabilizar el sistema.
TIEMPOTOTAL	Real	Tiempo total de la simulación.
dt	Real	Último paso de integración utilizado.
FORZADO	Entero	Variable que indica si se está usando el forzado.
CALCULAINVARIANTES	Entero	Variable que indica si se calculan los invariantes.
PID.error	Real	Último valor calculado del error en el PID para alcanzar el $k_{max}\eta$ deseado.
PID.deriv_error	Real	Último valor calculado de la derivada del error en el PID para alcanzar el $k_{max}\eta$ deseado.

Variable	**Tipo**	**Descripción**
PID.int_error	Real	Último valor calculado de la integral del error en el PID para alcanzar el $k_{max}\eta$ deseado.
PID.error_ant	Real	Valor del error en el paso anterior.
PID.K_P	Real	Constante de proporcionalidad del PID.
PID.K_I	Real	Constante de derivativa del PID.
PID.K_D	Real	Constante de Integral del PID.
Espectro.u_prima_cuadrado	Real	Valor de u'^2.
Espectro.EPSILON	Real	Valor de ϵ calculado a partir del espectro.
Espectro.L	Real	Valor de L calculado a partir del espectro.
Imagen.alto	Entero	Alto en píxeles de las imágenes que se generan.
Imagen.ancho	Entero	Ancho en píxeles de las imágenes que se generan.
Imagen.Divisiones_X	Entero	Número de décadas de k que se muestran en la gráfica del espectro.
Imagen.Divisiones_Y	Entero	Número de décadas de $E(k)$ que se muestran en la gráfica del espectro.
Imagen.MARGEN_X	Entero	Tamaño en píxeles del margen a los lados de las imágenes.
Imagen.MARGEN_Y	Entero	Tamaño en píxeles del margen superior e inferior de las imágenes.
Imagen.Puntos_Por_Division_X	Entero	Resolución en píxeles de cada década de k.
Imagen.Puntos_Por_Division_Y	Entero	Resolución en píxeles de cada década de $E(k)$.
Imagen.KETA_MAX	Real	Valor máximo que se muestra en la gráfica de $k_{max}\eta$.
Imagen.KETA_MIN	Real	Valor mínimo que se muestra en la gráfica de $k_{max}\eta$.
VX.Frec	Real	Componente X de velocidad en el instante actual.
VY.Frec	Real	Componente Y de velocidad en el instante actual.
VZ.Frec	Real	Componente Z de velocidad en el instante actual.
VVORTX.Frec	Real	Componente X de $\left(\widetilde{\mathbf{v}\times\omega}\right)_k$ en el instante actual.
VVORTY.Frec	Real	Componente Y de $\left(\widetilde{\mathbf{v}\times\omega}\right)_k$ en el instante actual.
VVORTZ.Frec	Real	Componente Z de $\left(\widetilde{\mathbf{v}\times\omega}\right)_k$ en el instante actual.
VX_nuevo.Frec	Real	Componente X de velocidad en el paso siguiente.
VY_nuevo.Frec	Real	Componente Y de velocidad en el paso siguiente.
VZ_nuevo.Frec	Real	Componente Z de velocidad en el paso siguiente.
VVORTX_ant.Frec	Real	Componente X de $\left(\widetilde{\mathbf{v}\times\omega}\right)_k$ en el instante anterior.
VVORTY_ant.Frec	Real	Componente Y de $\left(\widetilde{\mathbf{v}\times\omega}\right)_k$ en el instante anterior.
VVORTZ_ant.Frec	Real	Componente Z de $\left(\widetilde{\mathbf{v}\times\omega}\right)_k$ en el instante anterior.

Cuadro 7.1: Formato del fichero de estado.

7.2. Parámetros

Durante la ejecución del programa además de la evolución del campo de velocidades se van calculando algunos parámetros que describen características del estado. En el capítulo

2 se hace una breve introducción a los mismos, en esta sección se incluye una tabla de los valores obtenidos para distintas simulaciones:

N	**32**	**64**	**128**	**128**	**128**
ν	0,1	0,1	0,1	0,25	0,04
$K_{max}\eta$	1,396965	1,396395	1,466518	1,466388	1,456440
ϵ	11,60728	227,5969	3298,908	51563,77	217,0350
η	9,634238e-02	4,578344e-02	2,346429e-02	2,346221e-02	2,330304e-02
u'^2	5,320374	48,48671	291,9565	1907,654	67,51930
L	1,662018	1,767455	1,554745	1,486001	1,760550
λ	0,8809655	0,6613457	0,4341388	0,4247529	0,4308568
Re	38,33600	123,07213	265,65515	295,9327	361,6615
Re_λ	20,94586	53,85266	88,62102	84,58828	88,50889
$\frac{dt}{T_\eta}$	0,024829	0,020108	0,019825	0,024451	0,021076

Cuadro 7.2: Resultados de las simulaciones.

Como se puede ver en el cuadro 7.2 para un mismo valor de la viscosidad, aumentando N se alcanzan valores de Re mayores. Esto se debe a la siguiente relación entre los valores:

$$N \sim \frac{L}{\eta} \sim Re^{\frac{3}{4}}. \tag{7.1}$$

De la definición del número de Reynolds $Re = \frac{u'L}{\nu}$, como L es proporcional al tamaño del dominio espacial, vemos que la relación $\frac{u'}{\nu}$ permanece constante para un Re determinado. Este efecto se puede comprobar en las últimas columnas del cuadro 7.2 donde vemos tres simulaciones para $N = 128$ y distintos valores de ν. En el cuadro se comprueba que para distintos valores de ν el valor de u' se ajusta para mantener la razón aproximadamente constante lo que hace que las variaciones del valor de Re dependan muy poco de las de ν (en el cuadro las variaciones son de un 15 % aproximadamente respecto a la media). Este efecto también se puede observar en el valor de λ que operando con las ecuaciones 2.3 y 2.18 podemos expresar en función de L y la relación $\frac{u'}{\nu}$. Y consecuentemente en el valor de Re_λ que a su vez es el producto de λ y $\frac{u'}{\nu}$.

7.3. Evolución de $K_{max}\eta$

Uno de los resultados que nos ofrece el programa es una gráfica con la evolución de $K_{max}\eta$ en el último tiempo integral. Con esta gráfica podemos observar el funcionamiento del PID. En la figura 7.2 se puede ver la evolución de los instantes iniciales. Esta gráfica muestra como el PID va ajustando la inyección de energía de forma que en un tiempo integral ha conseguido que el sistema tenga un valor $K_{max}\eta$ muy próximo al deseado. El tiempo que tarda en estabilizarse el sistema en valores próximos al régimen permanente depende de las constantes del PID además de los parámetros propios de la simulación, siendo N un parámetro de gran influencia.

A continuación explicamos brevemente el funcionamiento de un PID y el significado de las constantes que lo regulan. Un controlador PID es aquel que regula un parámetro controlador (en nuestro caso la *viscosidad negativa*) en función del error cometido en otro parámetro

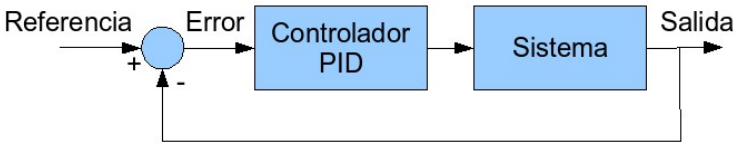

Figura 7.1: Esquema de un controlador PID.

controlado frente al resultado deseado. En la figura 7.1 se representa un esquema de su funcionamiento. Como se puede ver la entrada del controlador es una función del error $e(t)$ que se comete entre el parámetro del sistema que se controla y una referencia externa. El controlador calcula una señal de control que genera a partir de la siguiente expresión:

$$PID = K_P e(t) + K_I \int e(t)dt + K_D \frac{de(t)}{dt}. \tag{7.2}$$

La influencia de un incremento en alguno de los parámetros del controlador en la respuesta del sistema se puede resumir en la siguiente tabla:

Constante	**Tiempo de subida**	**Sobreoscilación**	**Tiempo de establecimiento**	**Error en régimen estacionario**
K_P	Decrece	Crece	Afecta poco	Decrece
K_I	Decrece	Crece	Crece	Lo elimina
K_D	Afecta poco	Decrece	Decrece	Afecta poco

Cuadro 7.3: Influencia de un incremento en las constantes de un controlador PID.

Experimentalmente se ha comprobado que las constantes del PID funcionan bien en los siguientes rangos: K_P entre 50 y 100, K_I un orden de magnitud menor que K_P, por ejemplo, $K_I = 5$. El efecto de K_D se ha comprobado que no ofrece ninguna ventaja y para ahorrar coste computacional se deja a cero.

Una de las características del código, es que se puede utilizar un filtro esférico que sea más estricto de lo que deseamos simular para poder estabilizar poco a poco el sistema, en la gráfica 7.3 se puede ver un ejemplo típico de evolución del filtro esférico en el que cuando el sistema está cerca de la zona estable el filtro crece y vuelve a estabilizarse hasta que se llega al tamaño deseado. Esta opción se introduce para poder simular cubos con valores de N elevados que son más difíciles de estabilizar.

7.4. Espectro

Para calcular el espectro se utiliza la siguiente fórmula obtenida de la bibliografía

$$E(n) = \frac{1}{2} \sum_{n-0,5<k<n+0,5} |\mathbf{v}(\mathbf{k})|^2 \tag{7.3}$$

En la figura 7.5 se representa la salida del programa, en ella se pueden ver cuatro líneas que se describen a continuación. La línea amarilla representa el espectro obtenido a partir de

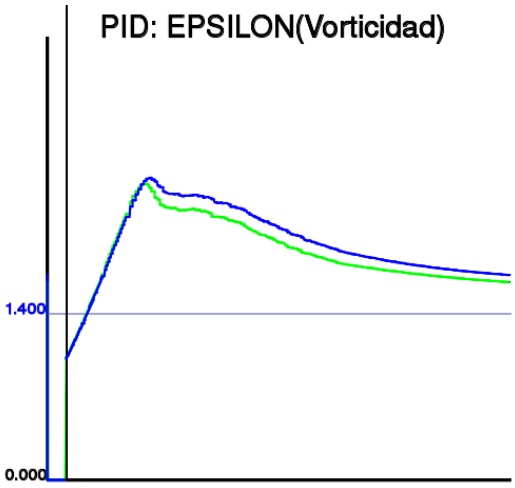

Figura 7.2: Evolución de $K_{max}\eta$ en los instantes iniciales.

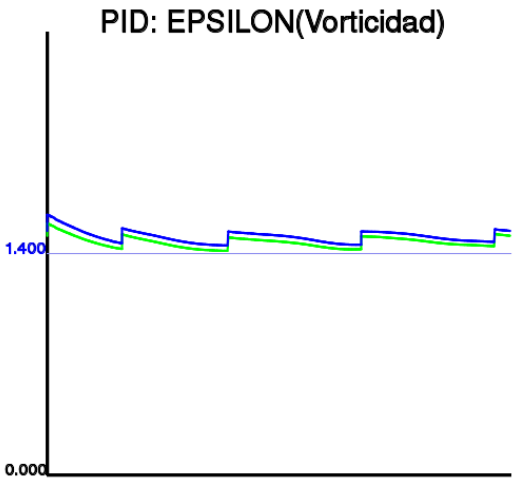

Figura 7.3: Evolución de $K_{max}\eta$. Cambios en el filtro esférico.

la expresión 2.20 con η tal que $K_{max}\eta = 1{,}4$ y L el real de la simulación en curso. La línea marrón representa la misma gráfica salvo que utiliza el valor real de η. La línea azul es el espectro de la simulación y la verde se corresponde con la siguiente expresión: $E(k)k^{\frac{5}{3}}\epsilon^{-\frac{2}{3}}$.

En la gráfica se puede observar que la zona donde se introduce la energía presenta ciertas variaciones respecto a la teórica, esto es debido al método con el que se introduce la energía en el sistema, se deja para estudios posteriores la implementación de otro tipo de sistemas.

7.5. Invariantes

Uno de los cálculos que mas tiempo de cómputo requiere es el cálculo de los invariantes y su representación gráfica, para su cálculo usamos las fórmulas que aparecen en el capítulo 2. Este cálculo da como resultado un archivo en el que se guardan las parejas de invariantes de cada punto del espacio en un instante dado, el número de datos es tan elevado que se recurre a una gráfica como la de la figura 7.6 que representa los isocontornos de probabilidad de que un punto caiga en una determinada zona de la gráfica para poder estudiarlos.

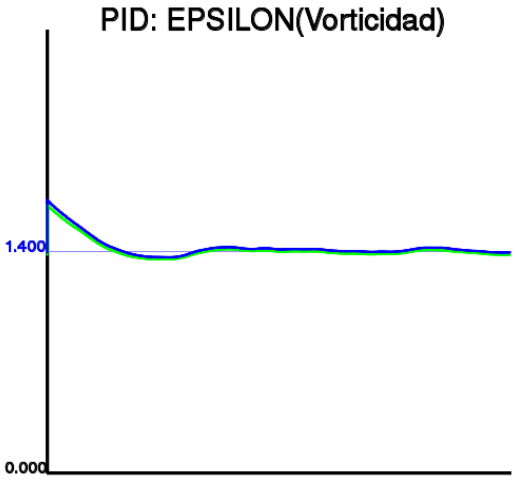

Figura 7.4: Evolución de $K_{max}\eta$. Entrada en régimen permanente.

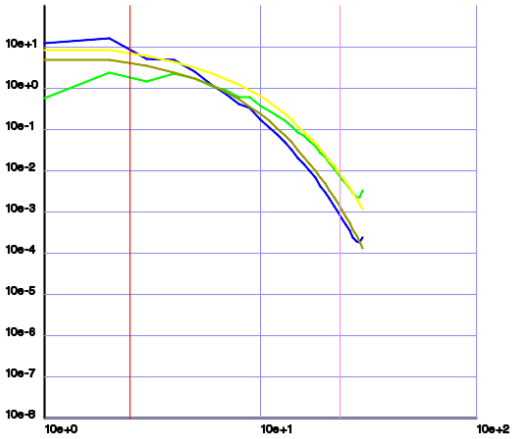

Figura 7.5: Espectro de una simulación con N=64 y $\nu = 0{,}1$.

Una representación alternativa es la de la figura 7.7 que se construye coloreando cada punto de la gráfica según el número de puntos del espacio que tienen sus invariantes en esa zona de la gráfica, de este modo si un punto de la gráfica no se corresponde con ninguna pareja de invariantes en el cubo se deja blanco. Según va correspondiendo a varios puntos del cubo se va cambiando su color de blanco a negro en varios tonos de gris, luego de negro a azul oscuro y finalmente de azul oscuro a celeste donde satura a partir de cierto número de puntos.

7.6. Errores numéricos

Para realizar una comparativa del error numérico, lo definimos como la diferencia máxima entre las componentes espectrales de la simulación y las mismas calculadas con un paso de integración de referencia mucho menor. El error relativo lo definimos como el error numérico dividido entre el valor de la simulación con el paso de integración de referencia.

En la gráfica 7.8 se pueden observar los errores numéricos relativos en función del paso integral. En esta gráfica podemos observar dos comportamientos. Para pasos grandes el error

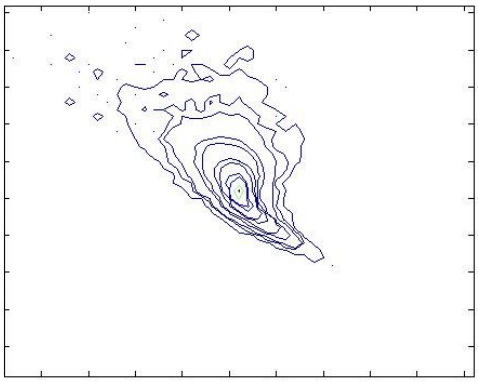

Figura 7.6: Isocontornos de la gráfica de invariantes con $N = 64$ y $\nu = 0{,}2$.

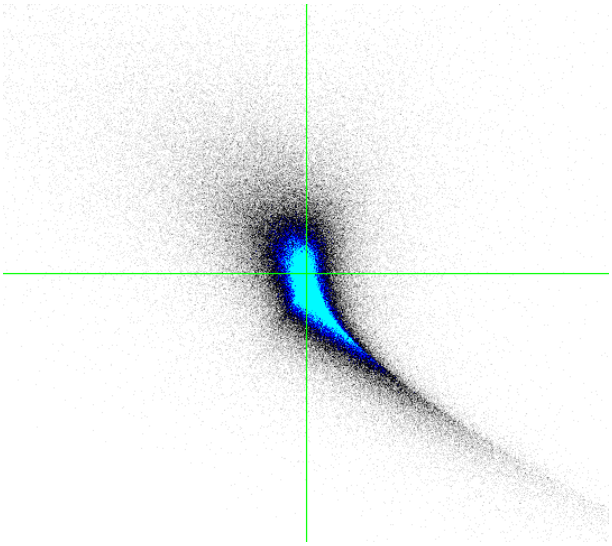

Figura 7.7: Ejemplo de invariantes con N=64 y $\nu = 0{,}2$.

relativo se hace enorme y vuelve los cálculos inservibles y para pasos muy pequeños los errores de redondeo hacen que la disminución del paso integral no redunde en precisión. Es por ello que el código utiliza un sistema que va ajustando el paso integral para mantenerse en un rango en el que los errores relativos sean suficientemente bajos y el número de pasos integrales sea el menor posible. Este rango se ha tomado partiendo de los resultados de la bibliografía en la que se indica que hay que mantener $\frac{dt}{T_\eta} \leq 0{,}025$ para que los errores relativos se mantengan bajos, por tanto se ha establecido el rango $0{,}015 \leq \frac{dt}{\tau_\eta} \leq 0{,}025$. Experimentalmente se ha comprobado que salir de ese rango no ofrece ventajas si se hace por abajo y vuelve el sistema inestable así como los cálculos erróneos para N suficientemente grande si se hace por arriba. La gráfica 7.8 muestra una comparativa realizada con un experimento con $N = 64$ y precisión simple, para pasos pequeños donde predominan los errores de redondeo se puede disminuir el error usando precisión doble en los cálculos numéricos.

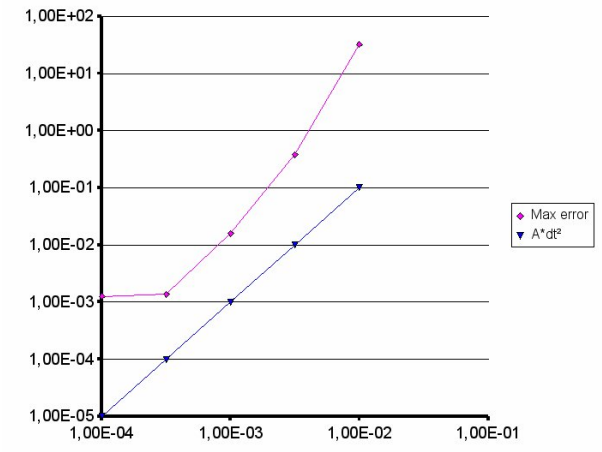

Figura 7.8: Comparativa del error relativo en función del tamaño del paso integral. Simulación con precisión simple, $N = 64$ y $\nu = 0{,}1$.

7.7. Paralelización

Los resultados de la paralelización pueden apreciarse en las gráficas 7.9 y 7.10. En estas gráficas se representa el tiempo necesario para realizar un mismo experimento en función del número de procesadores y su inverso que nos da una idea de la velocidad. El experimento de prueba consiste en la ejecución de mil pasos integrales con $N = 128$. En las gráficas se puede observar que el aumento del rendimiento no coincide con el número de procesadores (caso ideal), esto es debido a distintos motivos como las pérdidas por las comunicaciones y la parte del código que no se paraleliza. Por otra parte también se observa como para un número de procesadores potencia de 2 se aprecia un incremento del rendimiento relativo, esto es debido a la librería fftw que está optimizada para que el número de procesadores sea potencia de 2.

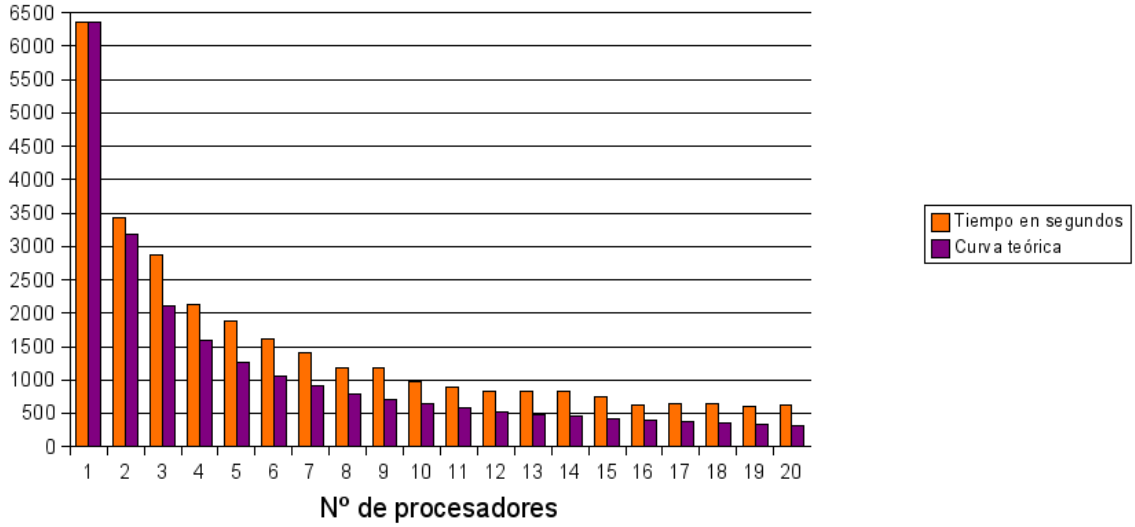

Figura 7.9: Tiempo empleado en una misma tarea en función del número de procesadores.

Se puede comprobar que a partir de cierto número de procesadores no hay un aumento

significativo del rendimiento e incluso si sigue aumentando el número empeora. Esto que en la gráfica 7.10 ocurre para 16 procesadores es un fenómeno habitual en las simulaciones en paralelo conocido como ley de Amdahl y se debe a la proporción de los intercambios de información frente al tiempo de uso de cada procesador. Así para un número pequeño de procesadores éstos consumen la mayor parte del tiempo en procesar los algoritmos pero según va aumentando el número de procesadores el intercambio de información entre los mismos aumenta hasta hacerse apreciable. Por tanto determinar el número óptimo de procesadores necesarios para una determinada simulación permite ahorrar recursos tanto económica como computacionalmete hablando.

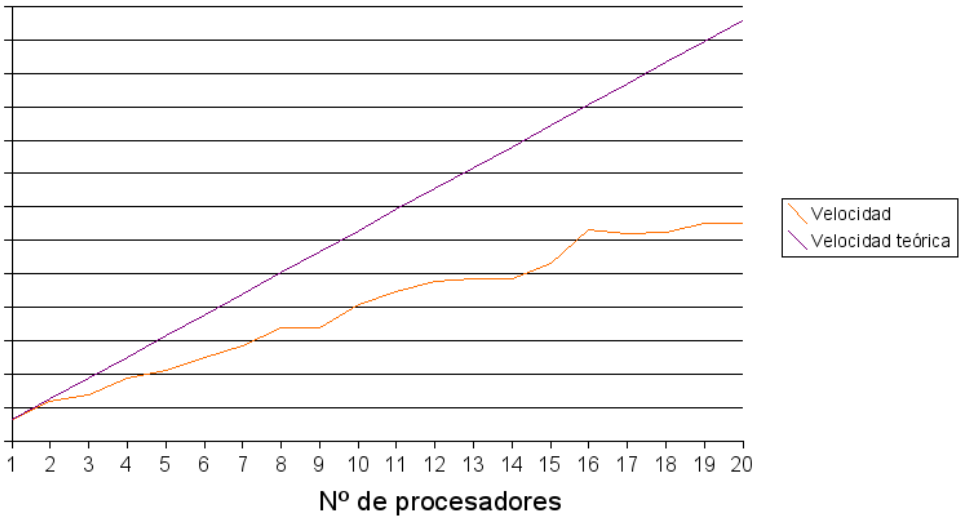

Figura 7.10: Velocidad de ejecución.

Capítulo 8

Conclusiones y futuras líneas de investigación

8.1. Conclusiones

En este proyecto se ha implementado un sistema que permite realizar simulaciones numéricas de turbulencias homogéneas e isótropas. Para ello debido a las necesidades computacionales se ha instalado un cluster de ordenadores para cálculo numérico y se ha diseñado un código capaz de usar todos los recursos del mismo.

El cluster está formado por diez nodos cada uno de ellos con dos procesadores AMD Opteron de 64 bits e interconectados mediante dos redes, una ethernet gigabit para usar un único sistema de ficheros en red mediante el protocolo NFS y otra red compuesta por tarjetas SCI que ofrecen una gran velocidad de interconexión y baja latencia lo que permite transferir la información que es necesario intercambiar entre los nodos con eficiencia.

El código se ha estructurado en archivos y funciones de forma que se puede variar cualquier característica del mismo para su estudio sin necesidad de conocer los algoritmos del resto de la implementación.

Para realizar las simulaciones se usa un método pseudoespectral en el que se integran las ecuaciones de Navier-Stokes utilizando el método de Adams-Bashforth de segundo orden salvo en el instante inicial que se usa un método de Runge-Kutta de segundo orden.

En la implentación se han usado distintas librerías de programación que permiten resolver algunas de las partes críticas del código con mayor eficiencia computacional, así en la paralelización del algoritmo FFT se ha usado el paquete FFTW que se adapta a las características propias de cada procesador para obtener mejor rendimiento. Para realizar el resto de las operaciones en paralelo se usa el estándar MPI a través del paquete NMPI que permite utilizar las tarjetas SCI para las comunicaciones.

Una de las características del código es que puede desactivar parte de los cálculos que se pueden realizar que no son estrictamente necesarios para aumentar la velocidad de la ejecución como puede ser la generación de gráficas en tiempo real, el guardar el estado en disco o el cálculo de los invariantes que pueden no ser necesarios para determinados estudios.

Existe además un sistema de configuración en tiempo real que permite variar los parámetros de la simulación así como pararla en cualquier momento.

Los resultados obtenidos son los esperados y son comparables a los de la bibliografía lo que permite asegurar su fiabilidad y uso en posibles estudios posteriores.

8.2. Futuras líneas de investigación

Actualmente las líneas de investigación usan modelos de las escalas pequeñas para realizar simulaciones con altos números de Reynolds reduciendo así el coste computacional. Existen modelos que permiten obtener la evolución del campo de velocidades, si bien no se ha conseguido aplicarlos con éxito al estudio de la dispersión de partículas.

Como posibles ampliaciones de este proyecto se podría añadir alguna de las siguientes líneas de investigación:

- Simulación de la evolución temporal de un conjunto de partículas en el interior de un flujo turbulento para su posible caracterización. Estudios de dispersión de partículas, de fluctuaciones de concentración de contaminantes o de fluctuaciones de temperatura.
- Búsqueda de algoritmos de integración temporal más precisos o con menor coste computacional.

Bibliografía

[1] A. Barrero y M. Pérez-Saborid (2005), *Fundamentos y aplicaciones de la Mecánica de Fluidos*. McGraw-Hill. ISBN 8448198905.

[2] G. K. Batchelor (1953), *The theory of homogeneous turbulence*. Cambridge Science Classics. ISBN 0-521-04117-1.

[3] Departamento de Matemática Aplicada II, apuntes de clase de la asignatura *Ampliación de Matemáticas* de Ingeniería de Telecomunicación. `http://www.personal.us.es/contreras/`

[4] Y. Couder, S. H. Davis, C. Garrett, P. Huerre, H. E. Huppert, J. Jiménez, P. F. Linden, M. E. McIntyre, H. K. Moffatt, T. J. Pedley y M. G. Worster (2000), *Perspectives in fluid dynamics. A collective introduction to current research*. Cambridge University Press. Cambridge. ISBN 0-521-78061-6.

[5] Debian GNU/Linux. `http://www.debian.org`

[6] Dolphin Interconnect Solutions Inc. `http://www.dolphinics.com`

[7] D. R. Durran (1990), *The third-order Adams-Bashforth method: An attractive Alternate to Leapfrog time differencing*. Monthly Weather Review vol. 119 pp. 702-720.

[8] Librería fftw. `http://www.fftw.org`.

[9] D. Gutiérrez (2004), *Stokesw, simulación de un flujo homogéneo e incompresible*. `davidi@eurus2.us.es`

[10] R. Haberman (1987), *Elementary applied partial differential equations*. Prentice-Hall International Editions.

[11] Librería imagemagick. `http://www.imagemagick.org`

[12] J. Jiménez, A. A. Wray, P. G. Saffman y R. S. Rogallo (1992), *The structure of intense vorticity in isotropic turbulence*. Journal of Fluid Mechanics vol. 255 pp. 65-90.

[13] B. W. Kernighan, D. M. Ritchie (1988), *The C programming language*. Englewood Cliffs, N.J. Prentice-Hall.

[14] MPICH2. `http://www-unix.mcs.anl.gov/mpi/m`

[15] NMPI. `http://www.nicevt.ru/research/nmpi/index.html?lang=en`

[16] K. Ogata (2003), *Ingeniería de control moderna*. Pearson Educación, D.L. Madrid. ISBN 84-2053-678-4.

[17] A. Ooi, J. Martín, J. Soria y M.S. Chon (1999). *A study of the evolution and characteristics of the invariants of the velocity-gradient tensor in isotropic turbulence*. J. Fluid Mech., vol. 381, pp. 141-174.

[18] E. O. Brigham (1988), *The fast Fourier transforms and its applications*. Prentice Hall Signal Processing Series. Englewood Cliffs, New Jersey. ISBN 0-13-307505-2.

[19] S. B. Pope (2000), *Turbulent Flows*. Cambridge University Press. ISBN 0-521-59886-9.

[20] J. G. Proakis y D. G. Manolakis (1998), *Tratamiento digital de señales. 3^a ed.*. Prentice Hall. Madrid. ISBN 84-8322-000-8.

[21] I. N. Sneddon (1951), *Fourier transforms*. Dover Publications, Inc. Nueva York. ISBN 0-486-68522-5(pbk.).

[22] A. Vincent y M. Meneguzzi (1991), *The spatial structure and statistical properties of homogeneous turbulence*. Journal of Fluid Mechanics vol. 225 pp. 1-20.

[23] L. P. Wang y M. R. Maxey (1993), *Settling velocity and concentration distribution of heavy particles in homogeneous isotropic turbulence*. Journal of Fluid Mechanics vol. 256 pp. 27-68.

Apéndice A

Código fuente

En este apéndice se incluye el código fuente del programa estructurado en los distintos archivos que lo componen. A continuación se describen brevemente los archivos que componen el proyecto:

- **navier_mpi.c:** En este archivo está la definición de la función *main*.
- **navier_mpi.h:** Fichero de cabecera de todos los demás en el que se definen las estructuras de datos y las declaraciones de funciones.
- **inicializacion_mpi.c:** En este fichero se definen las funciones de inicializacion y finalización en las que se reserva la memoria y libera.
- **integracion_mpi.c:** Fichero en el que se definen las funciones relacionadas con los pasos de integración.
- **vorticidad_mpi.c:** Fichero en el que se define la función que realiza los cálculos de $\left(\widetilde{\mathbf{u} \times \omega}\right)_k$.
- **espectro_mpi.c:** Fichero en el que se definen las funciones que calculan el espectro y los parámetros asociados.
- **invariantes_mpi.c:** Fichero en el que se define el cálculo de invariantes.
- **funciones_aux_mpi.c:** Fichero en el que se definen el PID y funciones auxiliares de menor interés.
- **imagenes_mpi.c:** Fichero que reune todos las funciones que generan las distintas gráficas de resultados.

```
/*************************************************************************
 *   Copyright (C) 2005 by Ignacio López Díaz                            *
 *   igna@us.es                                                          *
 *                                                                       *
 *   This program is free software; you can redistribute it and/or modify *
 *   it under the terms of the GNU General Public License as published by *
 *   the Free Software Foundation; either version 2 of the License, or   *
 *   (at your option) any later version.                                 *
 *                                                                       *
 *   This program is distributed in the hope that it will be useful,     *
 *   but WITHOUT ANY WARRANTY; without even the implied warranty of      *
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the       *
 *   GNU General Public License for more details.                        *
 *                                                                       *
 *   You should have received a copy of the GNU General Public License   *
 *   along with this program; if not, write to the                       *
 *   Free Software Foundation, Inc.,                                     *
 *   59 Temple Place - Suite 330, Boston, MA  02111-1307, USA.           *
 *************************************************************************/


#ifdef HAVE_CONFIG_H
#include <config.h>
#endif
#define PRINCIPAL

#include "navier_mpi.h"

int main(int argc, char *argv[])
{
    setlocale(LC_ALL,"es_ES@euro");

    //  Inicialización del sistema MPI
    if(MPI_Init(&argc, &argv)!=MPI_SUCCESS)
    {
        printf("Error: No se puede inicializar el MPI\n"); fflush(stdout);
        exit(1);
    }
    if(MPI_Comm_size(MPI_COMM_WORLD, &MPI.NUMERO_DE_PROCESOS_MPI)!=MPI_SUCCESS)
    {
        printf("Error: No se puede inicializar el MPI\n"); fflush(stdout);
        exit(1);
    }
    if(MPI_Comm_rank(MPI_COMM_WORLD, &MPI.PROCESO_MPI)!=MPI_SUCCESS)
    {
        printf("Error: No se puede inicializar el MPI\n"); fflush(stdout);
        exit(1);
    }

//  Identificamos el programa
    if(MPI.PROCESO_MPI==0)
    {
        printf("\n\nNavier-Stokes\n\nTamaño fftw_real: %d\n",sizeof(fftw_real));
        fflush(stdout);
    }
    MPI_Barrier(MPI_COMM_WORLD);

//  Comprobamos los parámetros y empezamos la integración
    if(argc==1)
    {
        Inicializacion(1);
        TIPOINT=ADAMS2CAD;
        Integracion(ADAMS2,1);
    }
    else    if(argc==2)
    {
        RecuperaEstado(argv[1],1);
        Inicializacion(0);
```



```
        RecuperaEstado(argv[1],0);

        TIPOINT=ADAMS2CAD;
        Integracion(ADAMS2,0);
    }
    else
    {
        if(MPI.PROCESO_MPI==0)
        {
            printf("\nError: El programa solo admite un parámetro opcional que es el
nombre del archivo a usar.\n");
            exit(1);
        }
    }

//  Finalizamos y salimos

    if(MPI.PROCESO_MPI==0)
    {
        printf("\n\nFin del programa.\n");
        fflush(stdout);
    }

    MPI_Barrier(MPI_COMM_WORLD);
    Finalizacion();
    MPI_Finalize();

    return EXIT_SUCCESS;
}
```

```
/*************************************************************************
 *   Copyright (C) 2005 by Ignacio López Díaz                            *
 *   igna@us.es                                                          *
 *                                                                       *
 *   This program is free software; you can redistribute it and/or modify *
 *   it under the terms of the GNU General Public License as published by *
 *   the Free Software Foundation; either version 2 of the License, or   *
 *   (at your option) any later version.                                 *
 *                                                                       *
 *   This program is distributed in the hope that it will be useful,     *
 *   but WITHOUT ANY WARRANTY; without even the implied warranty of      *
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the       *
 *   GNU General Public License for more details.                        *
 *                                                                       *
 *   You should have received a copy of the GNU General Public License   *
 *   along with this program; if not, write to the                       *
 *   Free Software Foundation, Inc.,                                     *
 *   59 Temple Place - Suite 330, Boston, MA  02111-1307, USA.           *
 *************************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <values.h>
#include <locale.h>
#include <math.h>
#include <string.h>
#include <wand/magick_wand.h>
#include <mpi.h>

#include <unistd.h>

#define CONIMAGENES

#ifndef PRINCIPAL
    #define EXT extern
#else
    #define EXT
#endif



#define BORRAR_PANTALLA "\033[1;1H\033[J"
#define NADA "\E[0;0m"
#define NEGRO "\E[30;0m"
#define ROJO "\E[31;47m"
#define FONDO_ROJO "\E[32;41m"
#define VERDE "\E[32;0m"
#define AMARILLO "\E[33;0m"
#define AZUL "\E[34;47m"
#define MAGENTA "\E[35;47m"
#define CELESTE "\E[36;0m"
#define BLANCO "\E[37;0m"

#define T_COMPROBACION_FILTRO   0.50
#define SOBRE_EXCITACION_MODOS 2.0

#ifdef PrecisionSimple
    #include <sfftw.h>
    #include <srfftw.h>

    #include <sfftw_mpi.h>
    #include <srfftw_mpi.h>
#else
    #include <fftw.h>
    #include <rfftw.h>

    #include <fftw_mpi.h>
    #include <rfftw_mpi.h>
#endif
```



```
#define KETA_VORTICIDAD 1
#define KETA_ESPECTRO   2
// En la siguiente línea se indica qué EPSILON se usa para los cálculos
#define TIPO_KETA   KETA_VORTICIDAD


/*Definición de funciones*/
void Inicializacion(int NUEVOINICIO);
void Finalizacion (void);
void Integracion(int tipo,int NUEVOINICIO);
void Integral_Adams2(void);
void Integral_Euler(void);
void Calcula_VVort(void);
void Calcula_V_1Vort_1(void);
void Paso_Integral_Euler(void);
void Paso_Integral_Runge_Kutta();

void Escribe_Estadisticas(void);


void Calcula_VISCOSIDADNEG(void);
void Calcula_Espectro(void);
void Calcula_Invariantes(void);

void Inicializa_Imagen_Invariantes(void);
void Genera_Imagen_Invariantes(void);
void Poner_Punto_Imagen_Invariantes(fftw_real determinante,fftw_real adjuntos);

void RecuperaEstado(char *Nombre,int cabecera);
void GuardaEstado(void);
void RecomponeDatos(void);

void Genera_Imagen_KETA(fftw_real KETA,fftw_real KETA_espectro);

void Calcula_Espectro_1D(void);

void Genera_Imagen_Espectro_Rodaja(void);
void Calcula_Espectro_Rodaja(void);

void LeeParametros (void);
void WebEstado(void);
fftw_real Modelo_E_k(fftw_real EPSILON_local,fftw_real K,fftw_real ETA_local,
fftw_real L_local);
fftw_real Numero_de_k(fftw_real K);

/*Variables globales accesibles a todo el código*/

EXT rfftwnd_mpi_plan Plan_mpi;
EXT rfftwnd_mpi_plan PlanInv_mpi;

EXT unsigned int N;
EXT unsigned int N3D;
EXT unsigned int N3D_DIV;
EXT unsigned int N3D_FREC;
EXT int Pasos;
EXT int GUARDAESTADO;
EXT int FORZADO;
EXT int CALCULAINVARIANTES;
EXT int CAMBIAR_dt;

EXT fftw_real AMPLITUD_INICIAL;
EXT fftw_real K_FILTRO;
EXT fftw_real Kcuadrado_FILTRO;
EXT fftw_real K_VISCOSIDAD;
EXT fftw_real K_VISCOSIDAD_CUAD;
```

```
EXT fftw_real VISCOSIDAD;
EXT fftw_real VISCOSIDADNEG;
EXT fftw_real EPSILON;
EXT fftw_real ETA;
EXT fftw_real KETA_ESTACIONARIA;
EXT fftw_real ETA_ESTACIONARIA;
EXT fftw_real EPSILON_ESTACIONARIO;

EXT fftw_real TIEMPOTOTAL;
EXT fftw_real dt;
EXT fftw_real dt_nuevo;


EXT fftw_real CURSOR_K;
EXT fftw_real CURSOR_E;

// PID
EXT struct
{
    fftw_real error;
    fftw_real deriv_error;
    fftw_real int_error;
    fftw_real error_ant;
    fftw_real K_P,K_I,K_D;
    fftw_real *KETA;
    fftw_real *KETA_espectro;
    int indice_KETA,tam_KETA;
    fftw_real Periodo_KETA;
} PID;

// Espectro
EXT struct
{
    double *Valores_Rodaja_MPI;
    double *Valores_Rodaja;
    int tam;
    fftw_real u_prima_cuadrado;
    fftw_real EPSILON;
    fftw_real L;
    fftw_real Lambda;
    fftw_real Re_lambda;
    fftw_real T;
} Espectro;

// Espacio de memoria auxiliar para aumentar la velocidad
EXT union
{
    fftw_real *Esp;
    fftw_complex *Frec;
} EspacioAux;

// Vectores del estado actual
EXT union
{
    fftw_real *Esp;
    fftw_complex *Frec;
} VX;
EXT union
{
    fftw_real *Esp;
    fftw_complex *Frec;
} VY;
EXT union
{
    fftw_real *Esp;
    fftw_complex *Frec;
} VZ;
// Variable para el cálculo de los invariantes
```



```
EXT struct {
    union
    {
        fftw_real *Esp;
        fftw_complex *Frec;
    } XX;
    union
    {
        fftw_real *Esp;
        fftw_complex *Frec;
    } XY;
    union
    {
        fftw_real *Esp;
        fftw_complex *Frec;
    } XZ;
    union
    {
        fftw_real *Esp;
        fftw_complex *Frec;
    } YX;
    union
    {
        fftw_real *Esp;
        fftw_complex *Frec;
    } YY;
    union
    {
        fftw_real *Esp;
        fftw_complex *Frec;
    } YZ;
    union
    {
        fftw_real *Esp;
        fftw_complex *Frec;
    } ZX;
    union
    {
        fftw_real *Esp;
        fftw_complex *Frec;
    } ZY;
    union
    {
        fftw_real *Esp;
        fftw_complex *Frec;
    } ZZ;
} Gradiente;

// Vectores del nuevo estado
EXT union
{
    fftw_real *Esp;
    fftw_complex *Frec;
} VX_nuevo;
EXT union
{
    fftw_real *Esp;
    fftw_complex *Frec;
} VY_nuevo;
EXT union
{
    fftw_real *Esp;
    fftw_complex *Frec;
} VZ_nuevo;

// V*Vorticidad del estado actual
EXT union
{
```

```
    fftw_real *Esp;
    fftw_complex *Frec;
} VVORTX;
EXT union
{
    fftw_real *Esp;
    fftw_complex *Frec;
} VVORTY;
EXT union
{
    fftw_real *Esp;
    fftw_complex *Frec;
} VVORTZ;

// V*Vorticidad del estado anterior
EXT union
{
    fftw_real *Esp;
    fftw_complex *Frec;
} VVORTX_ant;
EXT union
{
    fftw_real *Esp;
    fftw_complex *Frec;
} VVORTY_ant;
EXT union
{
    fftw_real *Esp;
    fftw_complex *Frec;
} VVORTZ_ant;

// Variables de estadísticas
EXT struct
{
    fftw_real u_Media_X;
    fftw_real u_Media_Y;
    fftw_real u_Media_Z;
    fftw_real u_Media_XX;
    fftw_real u_Media_YY;
    fftw_real u_Media_ZZ;
    fftw_real u_Media_XY;
    fftw_real u_Media_YZ;
    fftw_real u_Media_ZX;

    fftw_real omega_prima;
    fftw_real Media_u_Cuadrado;
    fftw_real Media_u_Cuadrado_ant;

    fftw_real KETA,KETA_espectro;
    fftw_real DIVMAX;
    fftw_real MaxEspectro;
    int NumCambiosDIVMAX;
    int cuantos;

} Estadisticas;

EXT struct
{
    unsigned char *buffer;
    int *buffer_MPI;
    int *buffer_MPI2;
    int alto;
    int ancho;
    int Divisiones_X,Divisiones_Y;
    int MARGEN_X,MARGEN_Y;
    int Puntos_Por_Division_X;
    int Puntos_Por_Division_Y;
    unsigned char **Cadena_X,**Cadena_Y;
```



```
    fftw_real KETA_MAX,KETA_MIN;
    fftw_real Escala_Invariantes_X,Escala_Invariantes_Y;
    DrawingWand *DW;
    PixelWand *PW;
    MagickWand *W;
} Imagen;

// Tipos de integración
#define ADAMS2  1
#define LEAPFROG    2
#define MAGAZENKOV  3
#define EULER       4

EXT char *ADAMS2CAD;
EXT char *LEAPFROGCAD;
EXT char *MAGAZENKOVCAD;
EXT char *EULERCAD;
EXT char *TIPOINT;

//  Inicialización del sistema MPI
EXT struct
{
    int NUMERO_DE_PROCESOS_MPI; /* Número de procesos */
    int PROCESO_MPI; /* Mi dirección: 0<=yo<=(nproc-1) */
    MPI_Status Estado;
    double DatosTX[9];
    double DatosRX[9];
} MPI;

// Tamaños de los vectores
EXT struct
{
    int NX,NY,NZ;
    int NX_offset,NY_offset;
    int TOTAL;
} TAM_LOCAL;
```

```
/*************************************************************************
 *   Copyright (C) 2005 by Ignacio López Díaz                            *
 *   igna@us.es                                                          *
 *                                                                       *
 *   This program is free software; you can redistribute it and/or modify *
 *   it under the terms of the GNU General Public License as published by *
 *   the Free Software Foundation; either version 2 of the License, or   *
 *   (at your option) any later version.                                 *
 *                                                                       *
 *   This program is distributed in the hope that it will be useful,     *
 *   but WITHOUT ANY WARRANTY; without even the implied warranty of      *
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the       *
 *   GNU General Public License for more details.                        *
 *                                                                       *
 *   You should have received a copy of the GNU General Public License   *
 *   along with this program; if not, write to the                       *
 *   Free Software Foundation, Inc.,                                     *
 *   59 Temple Place - Suite 330, Boston, MA  02111-1307, USA.           *
 *************************************************************************/

#include "navier_mpi.h"

void Inicializacion(int NUEVOINICIO)
{
    FILE *DATOS;
    int indice_Matriz=0;
    int indice_x,indice_z,indice_y;
    fftw_real X_x,Y_y,Z_z;
    double LeeDatos;
    int LeeCaracter;

    fftw_real aux_x,aux_y,aux_z;
    fftw_real p11,p12,p13,p21,p22,p23,p31,p32,p33,modulo_cuadrado_k;
//  Los valores actuales
    int z,y,x;

    GUARDAESTADO=1; //indica que se guarde el estado en disco


    CURSOR_K=23.5;
    CURSOR_E=0.0;
    Pasos=1;
    FORZADO=1;
    CAMBIAR_dt=0;
    CALCULAINVARIANTES=0;

    //Si es un nuevo inicio se piden los parámetros
    if(NUEVOINICIO==1)
    {
        //El proceso 0 pide los parámetros y se los envía al resto
        if(MPI.PROCESO_MPI==0)
        {
            printf("\nValor de N: ");
            z=scanf("%ud",&N);
            if(z==0)
            {
                printf("\nValor incorrecto");
                exit(1);
            }
        }
        MPI_Barrier(MPI_COMM_WORLD);
        MPI_Bcast ( &N, 1, MPI_INTEGER, 0, MPI_COMM_WORLD);
        if(MPI.PROCESO_MPI==0)
        {
            printf("\n");
            printf("\nValor de la viscosidad: ");
            z=scanf("%le",&LeeDatos);
            if(z==0)
```



```
            {
                printf("\nValor incorrecto");
                exit(1);
            }
        }
        MPI_Barrier(MPI_COMM_WORLD);
        MPI_Bcast ( &LeeDatos, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
        VISCOSIDAD=LeeDatos;

        if(MPI.PROCESO_MPI==0)
        {
            printf("\n");
            printf("\ndt: ");
            z=scanf("%le",&LeeDatos);
            if((z==0)||(LeeDatos==0.0))
            {
                printf("\nValor incorrecto");
                exit(1);
            }
            printf("\n");
        }
        MPI_Barrier(MPI_COMM_WORLD);
        MPI_Bcast ( &LeeDatos, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
        dt=LeeDatos;

        if(MPI.PROCESO_MPI==0)
        {
            printf("\nPaso fijo (s=1/n=0): ");
            fflush(stdin);
            z=scanf("%d",&LeeCaracter);

            if(LeeCaracter==1)
            {
                CAMBIAR_dt=-1;
            }
            else if(LeeCaracter==0)
            {
                CAMBIAR_dt=0;
            }
            else
            {
                printf("\nValor incorrecto.\n");
                exit(1);
            }
        }
        MPI_Barrier(MPI_COMM_WORLD);
        MPI_Bcast ( &CAMBIAR_dt, 1, MPI_INTEGER, 0, MPI_COMM_WORLD);


        AMPLITUD_INICIAL=10.0;

        if( (N/2.0-1.5) > (14.5))
            K_FILTRO=14.5;
        else
            K_FILTRO=(((fftw_real)N/2.0)-1.5);

        Kcuadrado_FILTRO=(K_FILTRO*K_FILTRO);
        K_VISCOSIDAD=2.5;
        K_VISCOSIDAD_CUAD=(K_VISCOSIDAD*K_VISCOSIDAD);
        N3D=(N*N*2*((N/2)+1));
        N3D_DIV= (N*N*N);
        N3D_FREC= (N*N*((N/2)+1));

        VISCOSIDADNEG=VISCOSIDAD;   //Inicialmente la viscosidad negativa se pone
igual a la positiva
        KETA_ESTACIONARIA=1.4;

        TIEMPOTOTAL=0.0;
```

```
        Estadisticas.Media_u_Cuadrado=0.0;

    //  Borramos el archivo de Divergencia
        if(MPI.PROCESO_MPI==0)
        {
            if ((DATOS=fopen("Divergencia.dat","w")) == NULL)
            {
                fprintf(stderr, "\nNo puedo abrir Divergencia.dat\n");
                exit(1);
            }
            else
                fclose(DATOS);
        }
        Estadisticas.NumCambiosDIVMAX=0;
    }

    ETA_ESTACIONARIA=KETA_ESTACIONARIA/K_FILTRO;
    EPSILON_ESTACIONARIO=VISCOSIDAD*VISCOSIDAD*VISCOSIDAD/pow(ETA_ESTACIONARIA,4.0);

    ADAMS2CAD="Adams-Bashforth (2° orden)";
    LEAPFROGCAD="Leapfrog (2° orden)";
    MAGAZENKOVCAD="Magazenkov (2° orden)";
    EULERCAD="Euler";

//  Inicializamos los planes de la fftw
    Plan_mpi = rfftw3d_mpi_create_plan(MPI_COMM_WORLD, N, N, N, FFTW_REAL_TO_COMPLEX,
FFTW_ESTIMATE);
    PlanInv_mpi = rfftw3d_mpi_create_plan(MPI_COMM_WORLD, N, N, N,
FFTW_COMPLEX_TO_REAL, FFTW_ESTIMATE);

//  Obtenemos el tamaño de las matrices locales
    rfftwnd_mpi_local_sizes(Plan_mpi, &TAM_LOCAL.NX, &TAM_LOCAL.NX_offset,
                            &TAM_LOCAL.NY,
                            &TAM_LOCAL.NY_offset,
                            &TAM_LOCAL.TOTAL);
//  Espacio de memoria auxiliar para aumentar la velocidad
    EspacioAux.Esp = fftw_malloc(sizeof(fftw_real) * TAM_LOCAL.TOTAL);
//  Valores actuales
    VX.Esp = fftw_malloc(sizeof(fftw_real) * TAM_LOCAL.TOTAL);
    VY.Esp = fftw_malloc(sizeof(fftw_real) * TAM_LOCAL.TOTAL);
    VZ.Esp = fftw_malloc(sizeof(fftw_real) * TAM_LOCAL.TOTAL);
    VVORTX.Esp = fftw_malloc(sizeof(fftw_real) * TAM_LOCAL.TOTAL);
    VVORTY.Esp = fftw_malloc(sizeof(fftw_real) * TAM_LOCAL.TOTAL);
    VVORTZ.Esp = fftw_malloc(sizeof(fftw_real) * TAM_LOCAL.TOTAL);

//  Los valores nuevos
    VX_nuevo.Esp = fftw_malloc(sizeof(fftw_real) * TAM_LOCAL.TOTAL);
    VY_nuevo.Esp = fftw_malloc(sizeof(fftw_real) * TAM_LOCAL.TOTAL);
    VZ_nuevo.Esp = fftw_malloc(sizeof(fftw_real) * TAM_LOCAL.TOTAL);

//  Los valores anteriores
    VVORTX_ant.Esp = fftw_malloc(sizeof(fftw_real) * TAM_LOCAL.TOTAL);
    VVORTY_ant.Esp = fftw_malloc(sizeof(fftw_real) * TAM_LOCAL.TOTAL);
    VVORTZ_ant.Esp = fftw_malloc(sizeof(fftw_real) * TAM_LOCAL.TOTAL);

//  Inicializamos el PID
    if(NUEVOINICIO==1)
    {
        PID.error=0.0;
        PID.deriv_error=0.0;
        PID.int_error=0.0;
        PID.error_ant=0.0;

        PID.K_P=50.0;
        PID.K_I=5.0;
        PID.K_D=0.0;
    }
```



```
        PID.VARIANZA_KETA=0.0;
//  Para la gráfica de KETA

        PID.tam_KETA=1000;
        PID.Periodo_KETA=0.0;
        PID.indice_KETA=0;
        PID.KETA=fftw_malloc(sizeof(fftw_real) * (PID.tam_KETA));
        PID.KETA_espectro=fftw_malloc(sizeof(fftw_real) * (PID.tam_KETA));

        if((PID.KETA==NULL)||(PID.KETA_espectro==NULL))
        {
            printf("\nError: No hay suficiente memoria disponible\n");
            exit(1);
        }

        for(z=0;z<PID.tam_KETA;z++)
        {
            PID.KETA[z]=0.0;
            PID.KETA_espectro[z]=0.0;
        }

//  Inicializamos el espectro en rodajas
        Espectro.tam=N/2;

        Espectro.Valores_Rodaja=fftw_malloc(sizeof(double) * (Espectro.tam+1));
        Espectro.Valores_Rodaja_MPI=fftw_malloc(sizeof(double) * (Espectro.tam+1));
        if((Espectro.Valores_Rodaja==NULL)||(Espectro.Valores_Rodaja_MPI==NULL))
        {
            printf("\nError: No hay suficiente memoria disponible\n");
            exit(1);
        }
        else
        {
            for(z=0;z<=Espectro.tam;z++)
            {
                Espectro.Valores_Rodaja[z]=0.0;
                Espectro.Valores_Rodaja_MPI[z]=0.0;
            }
        }
        Espectro.u_prima_cuadrado=0.0;

//  Reservamos memoria para los invariantes
        Gradiente.XX.Esp = fftw_malloc(sizeof(fftw_real) * TAM_LOCAL.TOTAL);
        Gradiente.XY.Esp = fftw_malloc(sizeof(fftw_real) * TAM_LOCAL.TOTAL);
        Gradiente.XZ.Esp = fftw_malloc(sizeof(fftw_real) * TAM_LOCAL.TOTAL);
        Gradiente.YX.Esp = fftw_malloc(sizeof(fftw_real) * TAM_LOCAL.TOTAL);
        Gradiente.YY.Esp = fftw_malloc(sizeof(fftw_real) * TAM_LOCAL.TOTAL);
        Gradiente.YZ.Esp = fftw_malloc(sizeof(fftw_real) * TAM_LOCAL.TOTAL);
        Gradiente.ZX.Esp = fftw_malloc(sizeof(fftw_real) * TAM_LOCAL.TOTAL);
        Gradiente.ZY.Esp = fftw_malloc(sizeof(fftw_real) * TAM_LOCAL.TOTAL);
        Gradiente.ZZ.Esp = fftw_malloc(sizeof(fftw_real) * TAM_LOCAL.TOTAL);

//  Comprobamos que no falte memoria
        if(
            (VX.Esp==NULL)||(VY.Esp==NULL)||(VZ.Esp==NULL)||
            (VVORTX.Esp==NULL)||(VVORTY.Esp==NULL)||(VVORTZ.Esp==NULL)||
            (VX_nuevo.Esp==NULL)||(VY_nuevo.Esp==NULL)||(VZ_nuevo.Esp==NULL)||
              (VVORTX_ant.Esp==NULL)||(VVORTY_ant.Esp==NULL)||(VVORTZ_ant.Esp==NULL)
              ||(EspacioAux.Esp==NULL)
        )
        {
            printf("\nError: No hay suficiente memoria disponible\n");
            exit(1);
        }
        if(
            (Gradiente.XX.Esp==NULL)||(Gradiente.XY.Esp==NULL)||(Gradiente.XZ.Esp==NULL)||
            (Gradiente.YX.Esp==NULL)||(Gradiente.YY.Esp==NULL)||(Gradiente.YZ.Esp==NULL)||
            (Gradiente.ZX.Esp==NULL)||(Gradiente.ZY.Esp==NULL)||(Gradiente.ZZ.Esp==NULL)
```

```
    )
    {
        printf("\nError: No hay suficiente memoria disponible para el cálculo del
gradiente\n");
        exit(1);
    }
//  Inicializamos las imagenes

    if(NUEVOINICIO==1)
    {
        Imagen.alto=480;
        Imagen.ancho=540;
        Imagen.Divisiones_Y=10;
        Imagen.MARGEN_X=50;
        Imagen.MARGEN_Y=30;

        Imagen.KETA_MAX=6.0;
        Imagen.KETA_MIN=0.0;
    }
    Imagen.Escala_Invariantes_X=0.01;
    Imagen.Escala_Invariantes_Y=0.01;
    Imagen.Divisiones_X=2+floor(log10(N/2));
    Imagen.Puntos_Por_Division_X=
(Imagen.ancho-2*Imagen.MARGEN_X)/(Imagen.Divisiones_X-1);
    Imagen.Puntos_Por_Division_Y=(Imagen.alto-2*Imagen.MARGEN_Y)/(Imagen.Divisiones_Y);

    Imagen.Cadena_X=malloc(sizeof(char *)*Imagen.Divisiones_X);
    Imagen.Cadena_Y=malloc(sizeof(char *)*Imagen.Divisiones_Y);

    for(z=0;z<Imagen.Divisiones_X;z++)
    {
        Imagen.Cadena_X[z]=malloc(sizeof(char)*10);
        sprintf((char *)Imagen.Cadena_X[z],"10e%+d",z);
    }

    for(z=0;z<Imagen.Divisiones_Y;z++)
    {
        Imagen.Cadena_Y[z]=malloc(sizeof(char)*10);
        sprintf((char *)Imagen.Cadena_Y[z],"10e%+d",1-z);
    }

    Imagen.buffer= fftw_malloc(sizeof(unsigned char) * 9 * Imagen.ancho * Imagen.alto);
    Imagen.buffer_MPI= fftw_malloc(sizeof(int) * Imagen.ancho * Imagen.alto);
    Imagen.buffer_MPI2= fftw_malloc(sizeof(int) * Imagen.ancho * Imagen.alto);

    if((Imagen.buffer==NULL)||(Imagen.buffer_MPI==NULL)||(Imagen.buffer_MPI2==NULL))
    {
        printf("\nError: No hay suficiente memoria disponible para la generación de
las imágenes\n");
        exit(1);
    }
    for (z=0;z<(sizeof(unsigned char) * 9 * Imagen.ancho *
Imagen.alto);z++)Imagen.buffer[z]=255;
    for (z=0;z< Imagen.ancho * Imagen.alto;z++)
    {
        Imagen.buffer_MPI[z]=0;
        Imagen.buffer_MPI2[z]=0;
    }


//  Hacemos un campo aleatorio
    srand(MPI.PROCESO_MPI);
    for(x=0;x<TAM_LOCAL.NX;x++)
        for(y=0;y<N;y++)
            for(z=0;z<N;z++)
            {
                if(rand()>RAND_MAX/2.0)
                    VX.Esp[z+(2*((N/2)+1))*(y+N*x)]=AMPLITUD_INICIAL*(
```



```
(1.0*rand())/(RAND_MAX+1.0));
                else
                    VX.Esp[z+(2*((N/2)+1))*(y+N*x)]=-AMPLITUD_INICIAL*(
(1.0*rand())/(RAND_MAX+1.0));

                if(rand()>RAND_MAX/2.0)
                    VY.Esp[z+(2*((N/2)+1))*(y+N*x)]=AMPLITUD_INICIAL*(
(1.0*rand())/(RAND_MAX+1.0));
                else
                    VY.Esp[z+(2*((N/2)+1))*(y+N*x)]=-AMPLITUD_INICIAL*(
(1.0*rand())/(RAND_MAX+1.0));

                if(rand()>RAND_MAX/2.0)
                    VZ.Esp[z+(2*((N/2)+1))*(y+N*x)]=AMPLITUD_INICIAL*(
(1.0*rand())/(RAND_MAX+1.0));
                else
                    VZ.Esp[z+(2*((N/2)+1))*(y+N*x)]=-AMPLITUD_INICIAL*(
(1.0*rand())/(RAND_MAX+1.0));

            }

//  Lo pasamos al espacio de Fourier
    rfftwnd_mpi(Plan_mpi, 1, VX.Esp, NULL, FFTW_TRANSPOSED_ORDER);
    rfftwnd_mpi(Plan_mpi, 1, VY.Esp, NULL, FFTW_TRANSPOSED_ORDER);
    rfftwnd_mpi(Plan_mpi, 1, VZ.Esp, NULL, FFTW_TRANSPOSED_ORDER);

//  Escalamos las frecuencias

    for(y=0;y<TAM_LOCAL.NY;y++)
    {
        for(x=0;x<N;x++)
        {
            for(z=0;z<((N/2)+1);z++)
            {
                indice_Matriz=z+((N/2)+1)*(x+N*y);

                VX.Fred[indice_Matriz].re/=N3D_DIV;
                VX.Fred[indice_Matriz].im/=N3D_DIV;

                VY.Fred[indice_Matriz].re/=N3D_DIV;
                VY.Fred[indice_Matriz].im/=N3D_DIV;

                VZ.Fred[indice_Matriz].re/=N3D_DIV;
                VZ.Fred[indice_Matriz].im/=N3D_DIV;
            }
        }
    }

//  Aseguramos que en frecuencia la divergencia sea cero

    for(y=0;y<TAM_LOCAL.NY;y++)
    {
        indice_y=((y+TAM_LOCAL.NY_offset)>=(N/2))?((y+TAM_LOCAL.NY_offset)-N):(y+TAM_LO
CAL.NY_offset);
        for(x=0;x<N;x++)
        {
            indice_x=(x>=(N/2))?(x-N):x;
            for(z=0;z<((N/2)+1);z++)
            {
                indice_z=(z>=(N/2))?(z-N):z;
                indice_Matriz=z+((N/2)+1)*(x+N*y);

                modulo_cuadrado_k=indice_z*indice_z+indice_x*indice_x+indice_y*indice_y
;

                p11=1.0-((indice_x*indice_x)/modulo_cuadrado_k);
                p12=-((indice_x*indice_y)/modulo_cuadrado_k);
                p13=-((indice_x*indice_z)/modulo_cuadrado_k);
```

```
                p21=-((indice_y*indice_x)/modulo_cuadrado_k);
                p22=1.0-((indice_y*indice_y)/modulo_cuadrado_k);
                p23=-((indice_y*indice_z)/modulo_cuadrado_k);
                p31=-((indice_z*indice_x)/modulo_cuadrado_k);
                p32=-((indice_z*indice_y)/modulo_cuadrado_k);
                p33=1.0-((indice_z*indice_z)/modulo_cuadrado_k);

                aux_x=VX.Fred[indice_Matriz].re;
                aux_y=VY.Fred[indice_Matriz].re;
                aux_z=VZ.Fred[indice_Matriz].re;
                VX.Fred[indice_Matriz].re=(p11*aux_x+p12*aux_y+p13*aux_z);
                VY.Fred[indice_Matriz].re=(p21*aux_x+p22*aux_y+p23*aux_z);
                VZ.Fred[indice_Matriz].re=(p31*aux_x+p32*aux_y+p33*aux_z);
                aux_x=VX.Fred[indice_Matriz].im;
                aux_y=VY.Fred[indice_Matriz].im;
                aux_z=VZ.Fred[indice_Matriz].im;
                VX.Fred[indice_Matriz].im=(p11*aux_x+p12*aux_y+p13*aux_z);
                VY.Fred[indice_Matriz].im=(p21*aux_x+p22*aux_y+p23*aux_z);
                VZ.Fred[indice_Matriz].im=(p31*aux_x+p32*aux_y+p33*aux_z);
            }
        }
    }

//  Borramos la media
    if(MPI.PROCESO_MPI==0)
    {
        VX.Fred[0].re=0.0;
        VX.Fred[0].im=0.0;

        VY.Fred[0].re=0.0;
        VY.Fred[0].im=0.0;

        VZ.Fred[0].re=0.0;
        VZ.Fred[0].im=0.0;
    }
//  Borramos las frecuencias altas
    for(y=0;y<TAM_LOCAL.NY;y++)
    {
        indice_y=((y+TAM_LOCAL.NY_offset)>=(N/2))?((y+TAM_LOCAL.NY_offset)-N):(y+TAM_LO
CAL.NY_offset);
        for(x=0;x<N;x++)
        {
            indice_x=(x>=(N/2))?(x-N):x;
            for(z=0;z<((N/2)+1);z++)
            {
                indice_z=(z>=(N/2))?(z-N):z;
                indice_Matriz=z+((N/2)+1)*(x+N*y);
                if((indice_z*indice_z+indice_y*indice_y+indice_x*indice_x)>((N/4)*(N/4)
))
                {
                    VX.Fred[indice_Matriz].re=0.0;
                    VX.Fred[indice_Matriz].im=0.0;

                    VY.Fred[indice_Matriz].re=0.0;
                    VY.Fred[indice_Matriz].im=0.0;

                    VZ.Fred[indice_Matriz].re=0.0;
                    VZ.Fred[indice_Matriz].im=0.0;
                }
            }
        }
    }

//  Sacamos los parámetros al archivo de salida
    if(MPI.PROCESO_MPI==0)
    {
        if ((DATOS=fopen("Parametros-salida.dat","w")) != NULL)
        {
```



```
            setlocale(LC_ALL,"es_ES@euro");
            fprintf(DATOS,"VISCOSIDAD:%f\n",VISCOSIDAD);
            fprintf(DATOS,"K_FILTRO:%f\n",K_FILTRO);
            fprintf(DATOS,"CURSOR_K:%f\n",CURSOR_K);
            fprintf(DATOS,"CURSOR_E:%f\n",CURSOR_E);

            fclose(DATOS);
        }
    }


    return;
}
void Finalizacion (void)
{
    //En esta función se liberan toda la memoria reservada
    int n;

    free(VX.Esp);free(VY.Esp);free(VZ.Esp);
    free(VX_nuevo.Esp);free(VY_nuevo.Esp);free(VZ_nuevo.Esp);
    free(VVORTX.Esp);free(VVORTY.Esp);free(VVORTZ.Esp);
    free(VVORTX_ant.Esp);free(VVORTY_ant.Esp);free(VVORTZ_ant.Esp);

    free(Gradiente.XX.Esp);free(Gradiente.XY.Esp);free(Gradiente.XZ.Esp);
    free(Gradiente.YX.Esp);free(Gradiente.YY.Esp);free(Gradiente.YZ.Esp);
    free(Gradiente.ZX.Esp);free(Gradiente.ZY.Esp);free(Gradiente.ZZ.Esp);

    free(EspacioAux.Esp);

    free(Imagen.buffer);

    for(n=0;n<Imagen.Divisiones_X;n++)
    {
        free(Imagen.Cadena_X[n]);

    }
    for(n=0;n<Imagen.Divisiones_Y;n++)
    {
        free(Imagen.Cadena_Y[n]);
    }
    free(Imagen.Cadena_X);
    free(Imagen.Cadena_Y);
    free(PID.KETA);
    free(PID.KETA_espectro);

    rfftwnd_mpi_destroy_plan(Plan_mpi);
    rfftwnd_mpi_destroy_plan(PlanInv_mpi);
}
```

```
/*************************************************************************
 *   Copyright (C) 2005 by Ignacio López Díaz                            *
 *   igna@us.es                                                          *
 *                                                                       *
 *   This program is free software; you can redistribute it and/or modify *
 *   it under the terms of the GNU General Public License as published by *
 *   the Free Software Foundation; either version 2 of the License, or   *
 *   (at your option) any later version.                                 *
 *                                                                       *
 *   This program is distributed in the hope that it will be useful,     *
 *   but WITHOUT ANY WARRANTY; without even the implied warranty of      *
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the       *
 *   GNU General Public License for more details.                        *
 *                                                                       *
 *   You should have received a copy of the GNU General Public License   *
 *   along with this program; if not, write to the                       *
 *   Free Software Foundation, Inc.,                                     *
 *   59 Temple Place - Suite 330, Boston, MA  02111-1307, USA.           *
 *************************************************************************/

#include "navier_mpi.h"

void Integracion(int tipo,int NUEVOINICIO)
{
    fftw_real *paux_x,*paux_y,*paux_z;

//  Si es un inicio nuevo
    if(NUEVOINICIO==1)
    {


        if(tipo==EULER)
        {
        //  Damos el primer paso con el método de Euler
            Paso_Integral_Euler();
        }
        else
        {
        //  Damos el primer paso con el método de Runge-Kutta
            Paso_Integral_Runge_Kutta();
        }

    //  Cambiamos el nuevo estado calculado (n+1) de velocidades al actual (n)
        paux_x=VX_nuevo.Esp;
        paux_y=VY_nuevo.Esp;
        paux_z=VZ_nuevo.Esp;
        VX_nuevo.Esp=VX.Esp;
        VY_nuevo.Esp=VY.Esp;
        VZ_nuevo.Esp=VZ.Esp;
        VX.Esp=paux_x;
        VY.Esp=paux_y;
        VZ.Esp=paux_z;

    //  Guardamos la vorticidad actual como la antigua
        paux_x=VVORTX_ant.Esp;
        paux_y=VVORTY_ant.Esp;
        paux_z=VVORTZ_ant.Esp;
        VVORTX_ant.Esp=VVORTX.Esp;
        VVORTY_ant.Esp=VVORTY.Esp;
        VVORTZ_ant.Esp=VVORTZ.Esp;
        VVORTX.Esp=paux_x;
        VVORTY.Esp=paux_y;
        VVORTZ.Esp=paux_z;

        if(tipo==EULER)
        {
        //  Damos el primer paso con el método de Euler
            Paso_Integral_Euler();
```



```
        }
        else
        {
        //  Damos el primer paso con el método de Runge-Kutta
            Paso_Integral_Runge_Kutta();
        }

    }

//  Continuamos con el resto de los métodos
    if(MPI.PROCESO_MPI==0)
    {
        printf("\nTipo de integración: %s.\n",TIPOINT);
    }
    switch (tipo)
    {
        case ADAMS2:
            Integral_Adams2();
        break;
        case LEAPFROG:
            printf("\nError: Método de integración no implementado aún.\n");
            exit(1);
        break;
        case MAGAZENKOV:
            printf("\nError: Método de integración no implementado aún.\n");
            exit(1);
        break;
        case EULER:
            printf("\nError: Método de integración no implementado aún.\n");
            exit(1);
        break;
        default:
            printf("\nError: No se conoce el método de integración.\n");
            exit(1);
    }

}


void Integral_Adams2(void)
{
    FILE *DATOS;
    fftw_real aux_x,aux_y,aux_z;
    fftw_real *paux_x,*paux_y,*paux_z;
    fftw_real p11,p12,p13,p21,p22,p23,p31,p32,p33,modulo_cuadrado_k;
    int x,z,y;
    int ultimo=0;
    int indice_x,indice_z,indice_y,indice_Matriz;

    fftw_real t=0.0,t_comprobacion_filtro=0.0;
    fftw_real exponencial;

    GUARDAESTADO=1;

//  Bucle temporal
    while(1)
    {
//      Comprueba si hay que aumentar el filtro esférico
        if( (t_comprobacion_filtro>T_COMPROBACION_FILTRO) && (K_FILTRO<
(((fftw_real)N/2.0)-1.5) ) &&
            (fabs( Estadisticas.KETA - KETA_ESTACIONARIA)<0.1) )
        {
            K_FILTRO+=1.0;
            if(K_FILTRO> (((fftw_real)N/2.0)-1.5) )
            {
                K_FILTRO = (((fftw_real)N/2.0)-1.5) ;
            }
            Kcuadrado_FILTRO=(K_FILTRO*K_FILTRO);
```

```
            t_comprobacion_filtro=0.0;
        }
        t=0.0;

        while(t<100.0*dt)
        {
        //  Cambiamos el nuevo estado calculado (n+1) de velocidades al actual (n)
            paux_x=VX_nuevo.Esp;
            paux_y=VY_nuevo.Esp;
            paux_z=VZ_nuevo.Esp;
            VX_nuevo.Esp=VX.Esp;
            VY_nuevo.Esp=VY.Esp;
            VZ_nuevo.Esp=VZ.Esp;
            VX.Esp=paux_x;
            VY.Esp=paux_y;
            VZ.Esp=paux_z;

        //  Guardamos la vorticidad actual como la antigua
            paux_x=VVORTX_ant.Esp;
            paux_y=VVORTY_ant.Esp;
            paux_z=VVORTZ_ant.Esp;
            VVORTX_ant.Esp=VVORTX.Esp;
            VVORTY_ant.Esp=VVORTY.Esp;
            VVORTZ_ant.Esp=VVORTZ.Esp;
            VVORTX.Esp=paux_x;
            VVORTY.Esp=paux_y;
            VVORTZ.Esp=paux_z;

        //  Comprobamos si hay que cambiar dt
            if(CAMBIAR_dt==1)
            {
                dt=dt_nuevo;
                //Aumentamos dt por adelantado para que funcionen las estadísticas del
método siguiente
                t+=dt;
                t_comprobacion_filtro+=dt;
                TIEMPOTOTAL+=dt;
            //  Damos el primer paso con el método de Runge-Kutta
                Paso_Integral_Runge_Kutta();

            //  Volvemos a cambiar las variables
                paux_x=VX_nuevo.Esp;
                paux_y=VY_nuevo.Esp;
                paux_z=VZ_nuevo.Esp;
                VX_nuevo.Esp=VX.Esp;
                VY_nuevo.Esp=VY.Esp;
                VZ_nuevo.Esp=VZ.Esp;
                VX.Esp=paux_x;
                VY.Esp=paux_y;
                VZ.Esp=paux_z;
            //  Guardamos la vorticidad actual como la antigua
                paux_x=VVORTX_ant.Esp;
                paux_y=VVORTY_ant.Esp;
                paux_z=VVORTZ_ant.Esp;
                VVORTX_ant.Esp=VVORTX.Esp;
                VVORTY_ant.Esp=VVORTY.Esp;
                VVORTZ_ant.Esp=VVORTZ.Esp;
                VVORTX.Esp=paux_x;
                VVORTY.Esp=paux_y;
                VVORTZ.Esp=paux_z;
            //  Ponemos el indicador a cero
                CAMBIAR_dt=0;
            }

        //  Calculamos la vorticidad del estado actual
            Calcula_VVort();

//          Calculamos los invariantes
```



```
            Calcula_Invariantes();

//          Calculamos la viscosidad negativa
            Calcula_VISCOSIDADNEG();

        //  Evaluamos la integral en cada k
            for(y=0;y<TAM_LOCAL.NY;y++)
            {
                indice_y=((y+TAM_LOCAL.NY_offset)>=(N/2))?((y+TAM_LOCAL.NY_offset)-N):(
y+TAM_LOCAL.NY_offset);
                for(x=0;x<N;x++)
                {
                    indice_x=(x>=(N/2))?(x-N):x;
                    for(z=0;z<((N/2)+1);z++)
                    {
                        indice_z=(z>=(N/2))?(z-N):z;
                        indice_Matriz=z+((N/2)+1)*(x+N*y);
                        modulo_cuadrado_k=indice_z*indice_z+indice_x*indice_x+indice_y*
indice_y;

                        if((modulo_cuadrado_k>0.0)&&(modulo_cuadrado_k<=(fftw_real)Kcua
drado_FILTRO))
                        {
                            p11=1.0-((fftw_real)(indice_x*indice_x)/modulo_cuadrado_k);
                            p12=-((fftw_real)(indice_x*indice_y)/modulo_cuadrado_k);
                            p13=-((fftw_real)(indice_x*indice_z)/modulo_cuadrado_k);
                            p21=-((fftw_real)(indice_y*indice_x)/modulo_cuadrado_k);
                            p22=1.0-((fftw_real)(indice_y*indice_y)/modulo_cuadrado_k);
                            p23=-((fftw_real)(indice_y*indice_z)/modulo_cuadrado_k);
                            p31=-((fftw_real)(indice_z*indice_x)/modulo_cuadrado_k);
                            p32=-((fftw_real)(indice_z*indice_y)/modulo_cuadrado_k);
                            p33=1.0-((fftw_real)(indice_z*indice_z)/modulo_cuadrado_k);

//                          Calculamos la exponencial con la viscosidad adecuada
                            if(modulo_cuadrado_k>K_VISCOSIDAD_CUAD)
                                exponencial=exp(-VISCOSIDAD*modulo_cuadrado_k*dt);
                            else
                            {
                                if(
                                    SOBRE_EXCITACION_MODOS*Modelo_E_k(EPSILON_ESTACIONA
RIO,sqrt(modulo_cuadrado_k),ETA_ESTACIONARIA,Espectro.L)/Numero_de_k(sqrt(modulo_cuadra
do_k))
                                    <
                                    (
                                    VX.Fred[indice_Matriz].re*VX.Fred[indice_Matriz].re
+VX.Fred[indice_Matriz].im*VX.Fred[indice_Matriz].im
                                    +
                                    VY.Fred[indice_Matriz].re*VY.Fred[indice_Matriz].re
+VY.Fred[indice_Matriz].im*VY.Fred[indice_Matriz].im
                                    +
                                    VZ.Fred[indice_Matriz].re*VZ.Fred[indice_Matriz].re
+VZ.Fred[indice_Matriz].im*VZ.Fred[indice_Matriz].im
                                    )
                                  )
                                {
                                    if(VISCOSIDADNEG<0.0)
                                        exponencial=exp(-VISCOSIDAD*modulo_cuadrado_k*d
t);
                                    else
                                        exponencial=exp(-VISCOSIDADNEG*modulo_cuadrado_
k*dt);
                                }
                                else
                                {
                                    exponencial=exp(-VISCOSIDADNEG*modulo_cuadrado_k*dt
);
                                }
                            }
```

```
            //              Calculamos primero las partes reales
                            aux_x=(3.0*VVORTX.Fred[indice_Matriz].re-exponencial*VVORTX
_ant.Fred[indice_Matriz].re);
                            aux_y=(3.0*VVORTY.Fred[indice_Matriz].re-exponencial*VVORTY
_ant.Fred[indice_Matriz].re);
                            aux_z=(3.0*VVORTZ.Fred[indice_Matriz].re-exponencial*VVORTZ
_ant.Fred[indice_Matriz].re);

                            VX_nuevo.Fred[indice_Matriz].re=exponencial*(VX.Fred[indice
_Matriz].re+((dt/2.0)*(p11*aux_x+p12*aux_y+p13*aux_z)));
                            VY_nuevo.Fred[indice_Matriz].re=exponencial*(VY.Fred[indice
_Matriz].re+((dt/2.0)*(p21*aux_x+p22*aux_y+p23*aux_z)));
                            VZ_nuevo.Fred[indice_Matriz].re=exponencial*(VZ.Fred[indice
_Matriz].re+((dt/2.0)*(p31*aux_x+p32*aux_y+p33*aux_z)));

            //              Continuamos con las partes imaginarias
                            aux_x=(3.0*VVORTX.Fred[indice_Matriz].im-exponencial*VVORTX
_ant.Fred[indice_Matriz].im);
                            aux_y=(3.0*VVORTY.Fred[indice_Matriz].im-exponencial*VVORTY
_ant.Fred[indice_Matriz].im);
                            aux_z=(3.0*VVORTZ.Fred[indice_Matriz].im-exponencial*VVORTZ
_ant.Fred[indice_Matriz].im);

                            VX_nuevo.Fred[indice_Matriz].im=exponencial*(VX.Fred[indice
_Matriz].im+((dt/2.0)*(p11*aux_x+p12*aux_y+p13*aux_z)));
                            VY_nuevo.Fred[indice_Matriz].im=exponencial*(VY.Fred[indice
_Matriz].im+((dt/2.0)*(p21*aux_x+p22*aux_y+p23*aux_z)));
                            VZ_nuevo.Fred[indice_Matriz].im=exponencial*(VZ.Fred[indice
_Matriz].im+((dt/2.0)*(p31*aux_x+p32*aux_y+p33*aux_z)));

//                          Corregimos la divergencia si es necesario
                            if(fabs(Estadisticas.DIVMAX)>1e-3)
                            {
                                aux_x=VX_nuevo.Fred[indice_Matriz].re;
                                aux_y=VY_nuevo.Fred[indice_Matriz].re;
                                aux_z=VZ_nuevo.Fred[indice_Matriz].re;
                                VX_nuevo.Fred[indice_Matriz].re=(p11*aux_x+p12*aux_y+p1
3*aux_z);
                                VY_nuevo.Fred[indice_Matriz].re=(p21*aux_x+p22*aux_y+p2
3*aux_z);
                                VZ_nuevo.Fred[indice_Matriz].re=(p31*aux_x+p32*aux_y+p3
3*aux_z);
                                aux_x=VX_nuevo.Fred[indice_Matriz].im;
                                aux_y=VY_nuevo.Fred[indice_Matriz].im;
                                aux_z=VZ_nuevo.Fred[indice_Matriz].im;
                                VX_nuevo.Fred[indice_Matriz].im=(p11*aux_x+p12*aux_y+p1
3*aux_z);
                                VY_nuevo.Fred[indice_Matriz].im=(p21*aux_x+p22*aux_y+p2
3*aux_z);
                                VZ_nuevo.Fred[indice_Matriz].im=(p31*aux_x+p32*aux_y+p3
3*aux_z);
                            }

                        }
                        else
                        {
            //              Calculamos primero las partes reales
                            VX_nuevo.Fred[indice_Matriz].re=0.0;
                            VY_nuevo.Fred[indice_Matriz].re=0.0;
                            VZ_nuevo.Fred[indice_Matriz].re=0.0;

            //              Continuamos con las partes imaginarias
                            VX_nuevo.Fred[indice_Matriz].im=0.0;
                            VY_nuevo.Fred[indice_Matriz].im=0.0;
                            VZ_nuevo.Fred[indice_Matriz].im=0.0;
                        }
                    }
```



```
                }
            }
//          Si ha sido necesario corregir la divergencia se indica un archivo en el
disco
            if(fabs(Estadisticas.DIVMAX)>1e-3)
            {
                if(MPI.PROCESO_MPI==0)
                {
                    if ((DATOS=fopen("Divergencia.dat","a")) == NULL)
                    {
                        fprintf(stderr, "\nNo puedo abrir Divergencia.dat\n");
                        exit(1);
                    }
                    fprintf(DATOS,"N= %d\tt= %e\tDIVMAX=
%e\n",N,t,Estadisticas.DIVMAX);
                    fclose(DATOS);
                }
                Estadisticas.NumCambiosDIVMAX++;

            }
//          Realizamos los cálculos con dt
            t+=dt;
            t_comprobacion_filtro+=dt;
            TIEMPOTOTAL+=dt;

//          Calculamos el espectro
            Calcula_Espectro_Rodaja();


//          Comprueba cuantos pasos tiene que dar
            if(Pasos>=0)
            {
                if(Pasos==0)
                {
                    if(MPI.PROCESO_MPI==0)printf("\nFin del proceso. Guardando el
estado.");
                    fflush(stdout);
                    x=GUARDAESTADO;
                    GUARDAESTADO=1;
                    GuardaEstado();
                    GUARDAESTADO=x;
                    if(CALCULAINVARIANTES)
                    {
                        if(MPI.PROCESO_MPI==0)printf("\nRecomponiendo el archivo de
invariantes.");
                        fflush(stdout);
                        RecomponeDatos();
                    }

                    if(MPI.PROCESO_MPI==0)
                    {
                        int valido=0;
            //          printf("\n¿Cuantos pasos doy? ");
                        while(valido!=1)
                        {
                            printf("\n¿Cuantos pasos doy (0=inf / -1=Salir)? ");
                            valido=scanf("%d",&Pasos);
                            fflush(stdin);
                            if(valido!=1)
                            {
                                printf("\nDebe introducir un número entero. ");
                                getchar();
                            }
                        }
                    }
                    MPI_Barrier(MPI_COMM_WORLD);
                    MPI_Bcast ( &Pasos, 1, MPI_INTEGER, 0, MPI_COMM_WORLD);
                    if(Pasos==-1)return;
```

```
                }
                Pasos--;
            }

            Escribe_Estadisticas(); //Estadísticas constantes
//          Sacamos el estado como una página web
            WebEstado();
//          Comprobamos si hay que cambiar el dt
            if( ( (dt/sqrt(VISCOSIDAD/Espectro.EPSILON)) >0.025  ) || (
(dt/sqrt(VISCOSIDAD/Espectro.EPSILON)) <0.015  ))
            {
                dt_nuevo=0.020*sqrt(VISCOSIDAD/Espectro.EPSILON);

                if(CAMBIAR_dt>=0)CAMBIAR_dt=1;
            }

//          Leemos los parámetros cada iteración
            LeeParametros();
        }

//      Guardamos el estado cada 100 iteraciones
        GuardaEstado();
    }

    return;
}




void Paso_Integral_Euler()
{
    fftw_real p11,p12,p13,p21,p22,p23,p31,p32,p33,modulo_cuadrado_k;
    int x,z,y;
    int indice_x,indice_z,indice_y,indice_Matriz;
    fftw_real exponencial;

//  Calculamos la vorticidad del estado actual
    Calcula_VVort();

//  Calculamos los invariantes
    Calcula_Invariantes();


//  Calculamos la viscosidad negativa
    Calcula_VISCOSIDADNEG();
//  Evaluamos la integral en cada k
    for(y=0;y<TAM_LOCAL.NY;y++)
    {
        indice_y=((y+TAM_LOCAL.NY_offset)>=(N/2))?((y+TAM_LOCAL.NY_offset)-N):(y+TAM_LO
CAL.NY_offset);
        for(x=0;x<N;x++)
        {
            indice_x=(x>=(N/2))?(x-N):x;
            for(z=0;z<((N/2)+1);z++)
            {
                indice_z=(z>=(N/2))?(z-N):z;
                indice_Matriz=z+((N/2)+1)*(x+N*y);
                modulo_cuadrado_k=indice_z*indice_z+indice_x*indice_x+indice_y*indice_y
;


                if((modulo_cuadrado_k>0)&&(modulo_cuadrado_k<=(fftw_real)Kcuadrado_FILT
RO))
                {
                    p11=1.0-((indice_x*indice_x)/modulo_cuadrado_k);
                    p12=-((indice_x*indice_y)/modulo_cuadrado_k);
                    p13=-((indice_x*indice_z)/modulo_cuadrado_k);
```



```
                    p21=-((indice_y*indice_x)/modulo_cuadrado_k);
                    p22=1.0-((indice_y*indice_y)/modulo_cuadrado_k);
                    p23=-((indice_y*indice_z)/modulo_cuadrado_k);
                    p31=-((indice_z*indice_x)/modulo_cuadrado_k);
                    p32=-((indice_z*indice_y)/modulo_cuadrado_k);
                    p33=1.0-((indice_z*indice_z)/modulo_cuadrado_k);

//                  Calculamos la exponencial con la viscosidad adecuada

                    if(modulo_cuadrado_k>K_VISCOSIDAD_CUAD)
                        exponencial=exp(-VISCOSIDAD*modulo_cuadrado_k*dt);
                    else
                    {
                        if(
                                                    SOBRE_EXCITACION_MODOS*Modelo_E_k(EPSILON
_ESTACIONARIO,sqrt(modulo_cuadrado_k),ETA_ESTACIONARIA,Espectro.L)/Numero_de_k(sqrt(mod
ulo_cuadrado_k))
                                                    <
                                                    (
                                                        VX.Fred[ indice_Matriz].re*VX.Frec
[ indice_Matriz].re+VX.Fred[ indice_Matriz].im*VX.Fred[ indice_Matriz].im
                                                        +
                                                        VY.Fred[ indice_Matriz].re*VY.Frec
[ indice_Matriz].re+VY.Fred[ indice_Matriz].im*VY.Fred[ indice_Matriz].im
                                                        +
                                                        VZ.Fred[ indice_Matriz].re*VZ.Frec
[ indice_Matriz].re+VZ.Fred[ indice_Matriz].im*VZ.Fred[ indice_Matriz].im
                                                    )
                          )
                        {
                            if(VISCOSIDADNEG<0.0)
                                exponencial=exp(-VISCOSIDAD*modulo_cuadrado_k*dt);
                            else
                                exponencial=exp(-VISCOSIDADNEG*modulo_cuadrado_k*dt);
                        }
                        else
                        {
                            exponencial=exp(-VISCOSIDADNEG*modulo_cuadrado_k*dt);
                        }
                    }

    //              Calculamos primero las partes reales

                    VX_nuevo.Fred[ indice_Matriz].re=VX.Fred[ indice_Matriz].re*exponenci
al+(dt*exponencial*(p11*VVORTX.Fred[ indice_Matriz].re+p12*VVORTY.Fred[ indice_Matriz].re
+p13*VVORTZ.Fred[ indice_Matriz].re));
                    VY_nuevo.Fred[ indice_Matriz].re=VY.Fred[ indice_Matriz].re*exponenci
al+(dt*exponencial*(p21*VVORTX.Fred[ indice_Matriz].re+p22*VVORTY.Fred[ indice_Matriz].re
+p23*VVORTZ.Fred[ indice_Matriz].re));
                    VZ_nuevo.Fred[ indice_Matriz].re=VZ.Fred[ indice_Matriz].re*exponenci
al+(dt*exponencial*(p31*VVORTX.Fred[ indice_Matriz].re+p32*VVORTY.Fred[ indice_Matriz].re
+p33*VVORTZ.Fred[ indice_Matriz].re));

    //              Continuamos con las partes imaginarias

                    VX_nuevo.Fred[ indice_Matriz].im=VX.Fred[ indice_Matriz].im*exponenci
al+(dt*exponencial*(p11*VVORTX.Fred[ indice_Matriz].im+p12*VVORTY.Fred[ indice_Matriz].im
+p13*VVORTZ.Fred[ indice_Matriz].im));
                    VY_nuevo.Fred[ indice_Matriz].im=VY.Fred[ indice_Matriz].im*exponenci
al+(dt*exponencial*(p21*VVORTX.Fred[ indice_Matriz].im+p22*VVORTY.Fred[ indice_Matriz].im
+p23*VVORTZ.Fred[ indice_Matriz].im));
                    VZ_nuevo.Fred[ indice_Matriz].im=VZ.Fred[ indice_Matriz].im*exponenci
al+(dt*exponencial*(p31*VVORTX.Fred[ indice_Matriz].im+p32*VVORTY.Fred[ indice_Matriz].im
+p33*VVORTZ.Fred[ indice_Matriz].im));
                }
                else
                {
    //              Calculamos primero las partes reales
```

```
                    VX_nuevo.Fred[indice_Matriz].re=0.0;
                    VY_nuevo.Fred[indice_Matriz].re=0.0;
                    VZ_nuevo.Fred[indice_Matriz].re=0.0;

    //              Continuamos con las partes imaginarias
                    VX_nuevo.Fred[indice_Matriz].im=0.0;
                    VY_nuevo.Fred[indice_Matriz].im=0.0;
                    VZ_nuevo.Fred[indice_Matriz].im=0.0;
                }
            }
        }
    }

    Escribe_Estadisticas();     //Estadísticas cada cierto tiempo

    Calcula_Espectro_Rodaja();

    return;
}


void Paso_Integral_Runge_Kutta()
{
    fftw_real p11,p12,p13,p21,p22,p23,p31,p32,p33,modulo_cuadrado_k;
    int x,z,y;
    int indice_x,indice_z,indice_y,indice_Matriz;
    fftw_real exponencial;

//  Calculamos la vorticidad del estado actual
    Calcula_VVort();

//  Calculamos la el estado intermedio u_1 por su vorticidad
    Calcula_V_1Vort_1();



//  Calculamos la viscosidad negativa
    Calcula_VISCOSIDADNEG();
//  Evaluamos la integral en cada k
    for(y=0;y<TAM_LOCAL.NY;y++)
    {
        indice_y=((y+TAM_LOCAL.NY_offset)>=(N/2))?((y+TAM_LOCAL.NY_offset)-N):(y+TAM_LO
CAL.NY_offset);
        for(x=0;x<N;x++)
        {
            indice_x=(x>=(N/2))?(x-N):x;
            for(z=0;z<((N/2)+1);z++)
            {
                indice_z=(z>=(N/2))?(z-N):z;
                indice_Matriz=z+((N/2)+1)*(x+N*y);
                modulo_cuadrado_k=indice_z*indice_z+indice_x*indice_x+indice_y*indice_y
;


                if((modulo_cuadrado_k>0)&&(modulo_cuadrado_k<=(fftw_real)Kcuadrado_FILT
RO))
                {
                    p11=1.0-((indice_x*indice_x)/modulo_cuadrado_k);
                    p12=-((indice_x*indice_y)/modulo_cuadrado_k);
                    p13=-((indice_x*indice_z)/modulo_cuadrado_k);
                    p21=-((indice_y*indice_x)/modulo_cuadrado_k);
                    p22=1.0-((indice_y*indice_y)/modulo_cuadrado_k);
                    p23=-((indice_y*indice_z)/modulo_cuadrado_k);
                    p31=-((indice_z*indice_x)/modulo_cuadrado_k);
                    p32=-((indice_z*indice_y)/modulo_cuadrado_k);
                    p33=1.0-((indice_z*indice_z)/modulo_cuadrado_k);

//                  Calculamos la exponencial con la viscosidad adecuada
```



```
                    if(modulo_cuadrado_k>K_VISCOSIDAD_CUAD)
                        exponencial=exp(-VISCOSIDAD*modulo_cuadrado_k*dt);
                    else
                    {
                        if(
                                                    SOBRE_EXCITACION_MODOS*Modelo_E_k(EPSILON
_ESTACIONARIO,sqrt(modulo_cuadrado_k),ETA_ESTACIONARIA,Espectro.L)/Numero_de_k(sqrt(mod
ulo_cuadrado_k))
                                                    <
                                                    (
                                                        VX.Fred[indice_Matriz].re*VX.Frec
[indice_Matriz].re+VX.Fred[indice_Matriz].im*VX.Fred[indice_Matriz].im
                                                        +
                                                        VY.Fred[indice_Matriz].re*VY.Frec
[indice_Matriz].re+VY.Fred[indice_Matriz].im*VY.Fred[indice_Matriz].im
                                                        +
                                                        VZ.Fred[indice_Matriz].re*VZ.Frec
[indice_Matriz].re+VZ.Fred[indice_Matriz].im*VZ.Fred[indice_Matriz].im
                                                    )
                          )
                        {
                            if(VISCOSIDADNEG<0.0)
                                exponencial=exp(-VISCOSIDAD*modulo_cuadrado_k*dt);
                            else
                                exponencial=exp(-VISCOSIDADNEG*modulo_cuadrado_k*dt);
                        }
                        else
                        {
                            exponencial=exp(-VISCOSIDADNEG*modulo_cuadrado_k*dt);
                        }
                    }

    //              Calculamos primero las partes reales

                    VX_nuevo.Fred[indice_Matriz].re=VX.Fred[indice_Matriz].re*exponenci
al+
                            ((dt/2.0)*exponencial*
                            (
                            p11*(VVORTX.Fred[indice_Matriz].re+VVORTX_ant.Fred[indice_M
atriz].re)
                            +
                            p12*(VVORTY.Fred[indice_Matriz].re+VVORTY_ant.Fred[indice_M
atriz].re)
                            +
                            p13*(VVORTZ.Fred[indice_Matriz].re+VVORTZ_ant.Fred[indice_M
atriz].re)
                            )
                            );
                    VY_nuevo.Fred[indice_Matriz].re=VY.Fred[indice_Matriz].re*exponenci
al+
                            ((dt/2.0)*exponencial*
                            (
                            p21*(VVORTX.Fred[indice_Matriz].re+VVORTX_ant.Fred[indice_M
atriz].re)
                            +
                            p22*(VVORTY.Fred[indice_Matriz].re+VVORTY_ant.Fred[indice_M
atriz].re)
                            +
                            p23*(VVORTZ.Fred[indice_Matriz].re+VVORTZ_ant.Fred[indice_M
atriz].re)
                            ));
                    VZ_nuevo.Fred[indice_Matriz].re=VZ.Fred[indice_Matriz].re*exponenci
al+
                            ((dt/2.0)*exponencial*
                            (
                            p31*(VVORTX.Fred[indice_Matriz].re+VVORTX_ant.Fred[indice_M
atriz].re)
                            +
```

```
                        p32*(VVORTY.Fred[indice_Matriz].re+VVORTY_ant.Fred[indice_M
atriz].re)
                        +
                        p33*(VVORTZ.Fred[indice_Matriz].re+VVORTZ_ant.Fred[indice_M
atriz].re)
                        ));


    //          Continuamos con las partes imaginarias

                    VX_nuevo.Fred[indice_Matriz].im=VX.Fred[indice_Matriz].im*exponenci
al+
                        ((dt/2.0)*exponencial*
                        (
                        p11*(VVORTX.Fred[indice_Matriz].im+VVORTX_ant.Fred[indice_M
atriz].im)
                        +
                        p12*(VVORTY.Fred[indice_Matriz].im+VVORTY_ant.Fred[indice_M
atriz].im)
                        +
                        p13*(VVORTZ.Fred[indice_Matriz].im+VVORTZ_ant.Fred[indice_M
atriz].im)
                        ));
                    VY_nuevo.Fred[indice_Matriz].im=VY.Fred[indice_Matriz].im*exponenci
al+
                        ((dt/2.0)*exponencial*
                        (
                        p21*(VVORTX.Fred[indice_Matriz].im+VVORTX_ant.Fred[indice_M
atriz].im)
                        +
                        p22*(VVORTY.Fred[indice_Matriz].im+VVORTY_ant.Fred[indice_M
atriz].im)
                        +
                        p23*(VVORTZ.Fred[indice_Matriz].im+VVORTZ_ant.Fred[indice_M
atriz].im)
                        ));
                    VZ_nuevo.Fred[indice_Matriz].im=VZ.Fred[indice_Matriz].im*exponenci
al+
                        ((dt/2.0)*exponencial*
                        (
                        p31*(VVORTX.Fred[indice_Matriz].im+VVORTX_ant.Fred[indice_M
atriz].im)
                        +
                        p32*(VVORTY.Fred[indice_Matriz].im+VVORTY_ant.Fred[indice_M
atriz].im)
                        +
                        p33*(VVORTZ.Fred[indice_Matriz].im+VVORTZ_ant.Fred[indice_M
atriz].im)
                        ));
                }
                else
                {
    //              Calculamos primero las partes reales
                    VX_nuevo.Fred[indice_Matriz].re=0.0;
                    VY_nuevo.Fred[indice_Matriz].re=0.0;
                    VZ_nuevo.Fred[indice_Matriz].re=0.0;

    //              Continuamos con las partes imaginarias
                    VX_nuevo.Fred[indice_Matriz].im=0.0;
                    VY_nuevo.Fred[indice_Matriz].im=0.0;
                    VZ_nuevo.Fred[indice_Matriz].im=0.0;
                }
            }
        }
    }

//  Calculamos los invariantes
    Calcula_Invariantes();
```



```
    Escribe_Estadisticas();     //Estadísticas cada cierto tiempo

    Calcula_Espectro_Rodaja();

    return;
}
```

```
/*****************************************************************************
 *   Copyright (C) 2005 by Ignacio López Díaz                                *
 *   igna@us.es                                                              *
 *                                                                           *
 *   This program is free software; you can redistribute it and/or modify    *
 *   it under the terms of the GNU General Public License as published by    *
 *   the Free Software Foundation; either version 2 of the License, or       *
 *   (at your option) any later version.                                     *
 *                                                                           *
 *   This program is distributed in the hope that it will be useful,         *
 *   but WITHOUT ANY WARRANTY; without even the implied warranty of          *
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the           *
 *   GNU General Public License for more details.                            *
 *                                                                           *
 *   You should have received a copy of the GNU General Public License       *
 *   along with this program; if not, write to the                           *
 *   Free Software Foundation, Inc.,                                         *
 *   59 Temple Place - Suite 330, Boston, MA  02111-1307, USA.               *
 *****************************************************************************/


#include "navier_mpi.h"

void Calcula_VVort(void)
{
//  En esta función se realiza el cálculo de la vorticidad y de algunos parámetros en
el dominio espacial

    int x,y,z;
    int indice_x,indice_z,indice_y,indice_Matriz;
    fftw_real aux_x,aux_y,aux_z;

//  En el siguiente bucle calculamos la vorticidad en frecuencia
    for(y=0;y<TAM_LOCAL.NY;y++)
    {
        indice_y=((y+TAM_LOCAL.NY_offset)>=(N/2))?((y+TAM_LOCAL.NY_offset)-N):(y+TAM_LO
CAL.NY_offset);
        for(x=0;x<N;x++)
        {
            indice_x=(x>=(N/2))?(x-N):x;
            for(z=0;z<((N/2)+1);z++)
            {
                indice_z=(z>=(N/2))?(z-N):z;
                indice_Matriz=z+((N/2)+1)*(x+N*y);

                VVORTX.Fred[indice_Matriz].im=((fftw_real)indice_y*VZ.Fred[indice_Matri
z].re)-((fftw_real)indice_z*VY.Fred[indice_Matriz].re);
                VVORTX.Fred[indice_Matriz].re=-(((fftw_real)indice_y*VZ.Fred[indice_Mat
riz].im)-((fftw_real)indice_z*VY.Fred[indice_Matriz].im));

                VVORTY.Fred[indice_Matriz].im=((fftw_real)indice_z*VX.Fred[indice_Matri
z].re)-((fftw_real)indice_x*VZ.Fred[indice_Matriz].re);
                VVORTY.Fred[indice_Matriz].re=-(((fftw_real)indice_z*VX.Fred[indice_Mat
riz].im)-((fftw_real)indice_x*VZ.Fred[indice_Matriz].im));

                VVORTZ.Fred[indice_Matriz].im=((fftw_real)indice_x*VY.Fred[indice_Matri
z].re)-((fftw_real)indice_y*VX.Fred[indice_Matriz].re);
                VVORTZ.Fred[indice_Matriz].re=-(((fftw_real)indice_x*VY.Fred[indice_Mat
riz].im)-((fftw_real)indice_y*VX.Fred[indice_Matriz].im));
            }
        }
    }

//  A continuación calculamos vvort

//  Primero pasamos la vorticidad al espacio normal haciendo la transformada inversa
    rfftwnd_mpi(PlanInv_mpi, 1, VVORTX.Esp, EspacioAux.Esp, FFTW_TRANSPOSED_ORDER);
    rfftwnd_mpi(PlanInv_mpi, 1, VVORTY.Esp, EspacioAux.Esp, FFTW_TRANSPOSED_ORDER);
```



```
    rfftwnd_mpi(PlanInv_mpi, 1, VVORTZ.Esp, EspacioAux.Esp, FFTW_TRANSPOSED_ORDER);

//  Copiamos el vector de velocidad en el nuevo para usarlo como variable auxiliar
    memcpy(VX_nuevo.Esp, VX.Esp, sizeof(fftw_real) * TAM_LOCAL.TOTAL);
    memcpy(VY_nuevo.Esp, VY.Esp, sizeof(fftw_real) * TAM_LOCAL.TOTAL);
    memcpy(VZ_nuevo.Esp, VZ.Esp, sizeof(fftw_real) * TAM_LOCAL.TOTAL);

//  Ahora pasamos la velocidad al espacio normal haciendo la transformada inversa
    rfftwnd_mpi(PlanInv_mpi, 1, VX_nuevo.Esp, EspacioAux.Esp, FFTW_TRANSPOSED_ORDER);
    rfftwnd_mpi(PlanInv_mpi, 1, VY_nuevo.Esp, EspacioAux.Esp, FFTW_TRANSPOSED_ORDER);
    rfftwnd_mpi(PlanInv_mpi, 1, VZ_nuevo.Esp, EspacioAux.Esp, FFTW_TRANSPOSED_ORDER);

//  Calculamos omega_prima, primero calculamos su cuadrado
    Estadisticas.omega_prima=0.0;
    Estadisticas.Media_u_Cuadrado_ant=Estadisticas.Media_u_Cuadrado;
    Estadisticas.Media_u_Cuadrado=0.0;

    for(x=0;x<TAM_LOCAL.NX;x++)
    {
        for(y=0;y<N;y++)
        {
            for(z=0;z<N;z++)
            {
                indice_Matriz=z+(2*((N/2)+1))*(y+N*x);

                Estadisticas.omega_prima+=VVORTX.Esp[indice_Matriz]*VVORTX.Esp[indice_M
atriz]+VVORTY.Esp[indice_Matriz]*VVORTY.Esp[indice_Matriz]+VVORTZ.Esp[indice_Matriz]*VV
ORTZ.Esp[indice_Matriz];

                Estadisticas.Media_u_Cuadrado+=VX_nuevo.Esp[indice_Matriz]*VX_nuevo.Esp
[indice_Matriz]+VY_nuevo.Esp[indice_Matriz]*VY_nuevo.Esp[indice_Matriz]+VZ_nuevo.Esp[in
dice_Matriz]*VZ_nuevo.Esp[indice_Matriz];
            }
        }
    }
//  Sumamos todos los datos
    MPI_Barrier(MPI_COMM_WORLD);

    MPI.DatosTX[0]=Estadisticas.omega_prima;
    MPI.DatosTX[1]=Estadisticas.Media_u_Cuadrado;

    MPI_Allreduce ( MPI.DatosTX,MPI.DatosRX, 2, MPI_DOUBLE,MPI_SUM, MPI_COMM_WORLD);

    Estadisticas.omega_prima=MPI.DatosRX[0]/N3D_DIV;
    Estadisticas.Media_u_Cuadrado=MPI.DatosRX[1]/N3D_DIV;

//  Calculamos EPSILON
    EPSILON=VISCOSIDAD*Estadisticas.omega_prima;

//  Finalizamos el cálculo de omega_prima
    Estadisticas.omega_prima=sqrt(Estadisticas.omega_prima);

//  Calculamos ETA
    ETA=pow(VISCOSIDAD*VISCOSIDAD*VISCOSIDAD/EPSILON,0.25);

//  Borramos las estadísticas
    Estadisticas.u_Media_XY=0.0;
    Estadisticas.u_Media_YZ=0.0;
    Estadisticas.u_Media_ZX=0.0;
    Estadisticas.u_Media_X=0.0;
    Estadisticas.u_Media_Y=0.0;
    Estadisticas.u_Media_Z=0.0;
    Estadisticas.u_Media_XX=0.0;
    Estadisticas.u_Media_YY=0.0;
    Estadisticas.u_Media_ZZ=0.0;

//  A continuación multiplicamos vectorialmente los coeficientes de ambas matrices
```

```
    for(x=0;x<TAM_LOCAL.NX;x++)
        for(y=0;y<N;y++)
            for(z=0;z<N;z++)
    {
        indice_Matriz=z+(2*((N/2)+1))*(y+N*x);
//              Calculamos el valor máximo
        Estadisticas.u_Media_X+=VX_nuevo.Esp[indice_Matriz];
        Estadisticas.u_Media_Y+=VY_nuevo.Esp[indice_Matriz];
        Estadisticas.u_Media_Z+=VZ_nuevo.Esp[indice_Matriz];

        Estadisticas.u_Media_XX+=VX_nuevo.Esp[indice_Matriz]*VX_nuevo.Esp[indice_Matriz
];
        Estadisticas.u_Media_YY+=VY_nuevo.Esp[indice_Matriz]*VY_nuevo.Esp[indice_Matriz
];
        Estadisticas.u_Media_ZZ+=VZ_nuevo.Esp[indice_Matriz]*VZ_nuevo.Esp[indice_Matriz
];

        Estadisticas.u_Media_XY+=VX_nuevo.Esp[indice_Matriz]*VY_nuevo.Esp[indice_Matriz
];
        Estadisticas.u_Media_YZ+=VY_nuevo.Esp[indice_Matriz]*VZ_nuevo.Esp[indice_Matriz
];
        Estadisticas.u_Media_ZX+=VZ_nuevo.Esp[indice_Matriz]*VX_nuevo.Esp[indice_Matriz
];

        aux_x=VVORTX.Esp[indice_Matriz];
        aux_y=VVORTY.Esp[indice_Matriz];
        aux_z=VVORTZ.Esp[indice_Matriz];

        VVORTX.Esp[indice_Matriz]=(VY_nuevo.Esp[indice_Matriz]*aux_z-VZ_nuevo.Esp[indic
e_Matriz]*aux_y);
        VVORTY.Esp[indice_Matriz]=(VZ_nuevo.Esp[indice_Matriz]*aux_x-VX_nuevo.Esp[indic
e_Matriz]*aux_z);
        VVORTZ.Esp[indice_Matriz]=(VX_nuevo.Esp[indice_Matriz]*aux_y-VY_nuevo.Esp[indic
e_Matriz]*aux_x);
    }

//  Sumamos los resultados de todos los nodos
    MPI_Barrier(MPI_COMM_WORLD);

    MPI.DatosTX[0]=Estadisticas.u_Media_XY;
    MPI.DatosTX[1]=Estadisticas.u_Media_YZ;
    MPI.DatosTX[2]=Estadisticas.u_Media_ZX;
    MPI.DatosTX[3]=Estadisticas.u_Media_X;
    MPI.DatosTX[4]=Estadisticas.u_Media_Y;
    MPI.DatosTX[5]=Estadisticas.u_Media_Z;
    MPI.DatosTX[6]=Estadisticas.u_Media_XX;
    MPI.DatosTX[7]=Estadisticas.u_Media_YY;
    MPI.DatosTX[8]=Estadisticas.u_Media_ZZ;

    MPI_Allreduce ( MPI.DatosTX,MPI.DatosRX, 9, MPI_DOUBLE,MPI_SUM, MPI_COMM_WORLD);

    Estadisticas.u_Media_XY=MPI.DatosRX[0]/N3D_DIV;
    Estadisticas.u_Media_YZ=MPI.DatosRX[1]/N3D_DIV;
    Estadisticas.u_Media_ZX=MPI.DatosRX[2]/N3D_DIV;
    Estadisticas.u_Media_X=MPI.DatosRX[3]/N3D_DIV;
    Estadisticas.u_Media_Y=MPI.DatosRX[4]/N3D_DIV;
    Estadisticas.u_Media_Z=MPI.DatosRX[5]/N3D_DIV;
    Estadisticas.u_Media_XX=sqrt(MPI.DatosRX[6]/N3D_DIV);
    Estadisticas.u_Media_YY=sqrt(MPI.DatosRX[7]/N3D_DIV);
    Estadisticas.u_Media_ZZ=sqrt(MPI.DatosRX[8]/N3D_DIV);

//  Finalmente pasamos al espacio de Fourier

    rfftwnd_mpi(Plan_mpi, 1, VVORTX.Esp, EspacioAux.Esp, FFTW_TRANSPOSED_ORDER);
    rfftwnd_mpi(Plan_mpi, 1, VVORTY.Esp, EspacioAux.Esp, FFTW_TRANSPOSED_ORDER);
    rfftwnd_mpi(Plan_mpi, 1, VVORTZ.Esp, EspacioAux.Esp, FFTW_TRANSPOSED_ORDER);

//  Escalamos las frecuencias
```



```
    for(y=0;y<TAM_LOCAL.NY;y++)
    {
        for(x=0;x<N;x++)
        {
            for(z=0;z<((N/2)+1);z++)
            {
                indice_Matriz=z+((N/2)+1)*(x+N*y);

                VVORTX.Fred[indice_Matriz].re/=N3D_DIV;
                VVORTX.Fred[indice_Matriz].im/=N3D_DIV;

                VVORTY.Fred[indice_Matriz].re/=N3D_DIV;
                VVORTY.Fred[indice_Matriz].im/=N3D_DIV;

                VVORTZ.Fred[indice_Matriz].re/=N3D_DIV;
                VVORTZ.Fred[indice_Matriz].im/=N3D_DIV;
            }
        }
    }

    return;
}

void Calcula_V_1Vort_1(void)
{
    int x,y,z;
    int indice_x,indice_z,indice_y,indice_Matriz;
    fftw_real p11,p12,p13,p21,p22,p23,p31,p32,p33,modulo_cuadrado_k,exponencial;
    fftw_real aux_x,aux_y,aux_z;

//  Calculamos primero V_1, Utilizamos como variable auxiliar VX_nuevo
    for(y=0;y<TAM_LOCAL.NY;y++)
    {
        indice_y=((y+TAM_LOCAL.NY_offset)>=(N/2))?((y+TAM_LOCAL.NY_offset)-N):(y+TAM_LO
CAL.NY_offset);
        for(x=0;x<N;x++)
        {
            indice_x=(x>=(N/2))?(x-N):x;
            for(z=0;z<((N/2)+1);z++)
            {
                indice_z=(z>=(N/2))?(z-N):z;
                indice_Matriz=z+((N/2)+1)*(x+N*y);
                modulo_cuadrado_k=(fftw_real)(indice_z*indice_z+indice_x*indice_x+indic
e_y*indice_y);

                if((modulo_cuadrado_k>0)&&(modulo_cuadrado_k<=(fftw_real)Kcuadrado_FILT
RO))
                {
```

```
                    p11=1.0-((indice_x*indice_x)/modulo_cuadrado_k);
                    p12=-((indice_x*indice_y)/modulo_cuadrado_k);
                    p13=-((indice_x*indice_z)/modulo_cuadrado_k);
                    p21=-((indice_y*indice_x)/modulo_cuadrado_k);
                    p22=1.0-((indice_y*indice_y)/modulo_cuadrado_k);
                    p23=-((indice_y*indice_z)/modulo_cuadrado_k);
                    p31=-((indice_z*indice_x)/modulo_cuadrado_k);
                    p32=-((indice_z*indice_y)/modulo_cuadrado_k);
                    p33=1.0-((indice_z*indice_z)/modulo_cuadrado_k);

//                  exponencial=exp(-VISCOSIDAD*modulo_cuadrado_k*dt);

                    VX_nuevo.Fred[indice_Matriz].re=(
VX.Fred[indice_Matriz].re+dt*(p11*VVORTX.Fred[indice_Matriz].re+p12*VVORTY.Frec[indice_
Matriz].re+p13*VVORTZ.Fred[indice_Matriz].re));

                    VX_nuevo.Fred[indice_Matriz].im=(
VX.Fred[indice_Matriz].im+dt*(p11*VVORTX.Fred[indice_Matriz].im+p12*VVORTY.Frec[indice_
Matriz].im+p13*VVORTZ.Fred[indice_Matriz].im));

                    VY_nuevo.Fred[indice_Matriz].re=(
VY.Fred[indice_Matriz].re+dt*(p21*VVORTX.Fred[indice_Matriz].re+p22*VVORTY.Frec[indice_
Matriz].re+p23*VVORTZ.Fred[indice_Matriz].re));

                    VY_nuevo.Fred[indice_Matriz].im=(
VY.Fred[indice_Matriz].im+dt*(p21*VVORTX.Fred[indice_Matriz].im+p22*VVORTY.Frec[indice_
Matriz].im+p23*VVORTZ.Fred[indice_Matriz].im));

                    VZ_nuevo.Fred[indice_Matriz].re=(
VZ.Fred[indice_Matriz].re+dt*(p31*VVORTX.Fred[indice_Matriz].re+p32*VVORTY.Frec[indice_
Matriz].re+p33*VVORTZ.Fred[indice_Matriz].re));

                    VZ_nuevo.Fred[indice_Matriz].im=(
VZ.Fred[indice_Matriz].im+dt*(p31*VVORTX.Fred[indice_Matriz].im+p32*VVORTY.Frec[indice_
Matriz].im+p33*VVORTZ.Fred[indice_Matriz].im));
                }
                else
                {
        //              Calculamos primero las partes reales
                    VX_nuevo.Fred[indice_Matriz].re=0.0;
                    VY_nuevo.Fred[indice_Matriz].re=0.0;
                    VZ_nuevo.Fred[indice_Matriz].re=0.0;

        //              Continuamos con las partes imaginarias
                    VX_nuevo.Fred[indice_Matriz].im=0.0;
                    VY_nuevo.Fred[indice_Matriz].im=0.0;
                    VZ_nuevo.Fred[indice_Matriz].im=0.0;
                }

            }
        }
    }

//  {
//      fftw_real *paux_x,*paux_y,*paux_z;
//      paux_x=VX_nuevo.Esp;
//      paux_y=VY_nuevo.Esp;
//      paux_z=VZ_nuevo.Esp;
//      VX_nuevo.Esp=VX.Esp;
//      VY_nuevo.Esp=VY.Esp;
//      VZ_nuevo.Esp=VZ.Esp;
//      VX.Esp=paux_x;
//      VY.Esp=paux_y;
//      VZ.Esp=paux_z;
//      if(MPI.PROCESO_MPI==0){printf("\n\nCalculando
euler\n\n");fflush(stdout);}getchar();MPI_Barrier(MPI_COMM_WORLD);
//      Paso_Integral_Euler();
//      if(MPI.PROCESO_MPI==0){printf("\n\nFin
```



```
euler\n\n");fflush(stdout);}getchar();MPI_Barrier(MPI_COMM_WORLD);
//      Finalizacion();
//      exit(3);
//  }
//  En el siguiente bucle calculamos la vorticidad en frecuencia de V_1
    for(y=0;y<TAM_LOCAL.NY;y++)
    {
        indice_y=((y+TAM_LOCAL.NY_offset)>=(N/2))?((y+TAM_LOCAL.NY_offset)-N):(y+TAM_LO
CAL.NY_offset);
        for(x=0;x<N;x++)
        {
            indice_x=(x>=(N/2))?(x-N):x;
            for(z=0;z<((N/2)+1);z++)
            {
                indice_z=(z>=(N/2))?(z-N):z;
                indice_Matriz=z+((N/2)+1)*(x+N*y);

                VVORTX_ant.Frec[indice_Matriz].im=((fftw_real)indice_y*VZ_nuevo.Fred[in
dice_Matriz].re)-((fftw_real)indice_z*VY_nuevo.Fred[indice_Matriz].re);
                VVORTX_ant.Fred[indice_Matriz].re=-(((fftw_real)indice_y*VZ_nuevo.Fred[
indice_Matriz].im)-((fftw_real)indice_z*VY_nuevo.Fred[indice_Matriz].im));

                VVORTY_ant.Frec[indice_Matriz].im=((fftw_real)indice_z*VX_nuevo.Fred[in
dice_Matriz].re)-((fftw_real)indice_x*VZ_nuevo.Fred[indice_Matriz].re);
                VVORTY_ant.Fred[indice_Matriz].re=-(((fftw_real)indice_z*VX_nuevo.Fred[
indice_Matriz].im)-((fftw_real)indice_x*VZ_nuevo.Fred[indice_Matriz].im));

                VVORTZ_ant.Frec[indice_Matriz].im=((fftw_real)indice_x*VY_nuevo.Fred[in
dice_Matriz].re)-((fftw_real)indice_y*VX_nuevo.Fred[indice_Matriz].re);
                VVORTZ_ant.Fred[indice_Matriz].re=-(((fftw_real)indice_x*VY_nuevo.Fred[
indice_Matriz].im)-((fftw_real)indice_y*VX_nuevo.Fred[indice_Matriz].im));
            }
        }
    }



//  A continuación calculamos vvort_1

//  Primero pasamos la vorticidad al espacio normal haciendo la transformada inversa
    rfftwnd_mpi(PlanInv_mpi, 1, VVORTX_ant.Esp, EspacioAux.Esp, FFTW_TRANSPOSED_ORDER);
    rfftwnd_mpi(PlanInv_mpi, 1, VVORTY_ant.Esp, EspacioAux.Esp, FFTW_TRANSPOSED_ORDER);
    rfftwnd_mpi(PlanInv_mpi, 1, VVORTZ_ant.Esp, EspacioAux.Esp, FFTW_TRANSPOSED_ORDER);


//  Ahora pasamos la velocidad al espacio normal haciendo la transformada inversa
    rfftwnd_mpi(PlanInv_mpi, 1, VX_nuevo.Esp, EspacioAux.Esp, FFTW_TRANSPOSED_ORDER);
    rfftwnd_mpi(PlanInv_mpi, 1, VY_nuevo.Esp, EspacioAux.Esp, FFTW_TRANSPOSED_ORDER);
    rfftwnd_mpi(PlanInv_mpi, 1, VZ_nuevo.Esp, EspacioAux.Esp, FFTW_TRANSPOSED_ORDER);



//  A continuación multiplicamos vectorialmente los coeficientes de ambas matrices
//  Estadisticas.VALMAX=0.0;
    for(x=0;x<TAM_LOCAL.NX;x++)
        for(y=0;y<N;y++)
            for(z=0;z<N;z++)
    {
        indice_Matriz=z+(2*((N/2)+1))*(y+N*x);
//              Calculamos el valor máximo
        aux_x=VVORTX_ant.Esp[indice_Matriz];
        aux_y=VVORTY_ant.Esp[indice_Matriz];
        aux_z=VVORTZ_ant.Esp[indice_Matriz];

        VVORTX_ant.Esp[indice_Matriz]=(VY_nuevo.Esp[indice_Matriz]*aux_z-VZ_nuevo.Esp[i
ndice_Matriz]*aux_y);
        VVORTY_ant.Esp[indice_Matriz]=(VZ_nuevo.Esp[indice_Matriz]*aux_x-VX_nuevo.Esp[i
ndice_Matriz]*aux_z);
```

```
        VVORTZ_ant.Esp[ indice_Matriz] =(VX_nuevo.Esp[ indice_Matriz]*aux_y-VY_nuevo.Esp[ i
ndice_Matriz]*aux_x);
    }

    MPI_Barrier(MPI_COMM_WORLD);



//  Finalmente pasamos al espacio de Fourier

    rfftwnd_mpi(Plan_mpi, 1, VVORTX_ant.Esp, EspacioAux.Esp, FFTW_TRANSPOSED_ORDER);
    rfftwnd_mpi(Plan_mpi, 1, VVORTY_ant.Esp, EspacioAux.Esp, FFTW_TRANSPOSED_ORDER);
    rfftwnd_mpi(Plan_mpi, 1, VVORTZ_ant.Esp, EspacioAux.Esp, FFTW_TRANSPOSED_ORDER);

//  Escalamos las frecuencias
    for(y=0;y<TAM_LOCAL.NY;y++)
    {
        for(x=0;x<N;x++)
        {
            for(z=0;z<((N/2)+1);z++)
            {
                indice_Matriz=z+((N/2)+1)*(x+N*y);

                VVORTX_ant.Frec[ indice_Matriz].re/=N3D_DIV;
                VVORTX_ant.Frec[ indice_Matriz].im/=N3D_DIV;

                VVORTY_ant.Frec[ indice_Matriz].re/=N3D_DIV;
                VVORTY_ant.Frec[ indice_Matriz].im/=N3D_DIV;

                VVORTZ_ant.Frec[ indice_Matriz].re/=N3D_DIV;
                VVORTZ_ant.Frec[ indice_Matriz].im/=N3D_DIV;
            }
        }
    }

    return;
}
```

```
/*************************************************************************
 *   Copyright (C) 2005 by Ignacio López Díaz                            *
 *   igna@us.es                                                          *
 *                                                                       *
 *   This program is free software; you can redistribute it and/or modify *
 *   it under the terms of the GNU General Public License as published by *
 *   the Free Software Foundation; either version 2 of the License, or   *
 *   (at your option) any later version.                                 *
 *                                                                       *
 *   This program is distributed in the hope that it will be useful,     *
 *   but WITHOUT ANY WARRANTY; without even the implied warranty of      *
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the       *
 *   GNU General Public License for more details.                        *
 *                                                                       *
 *   You should have received a copy of the GNU General Public License   *
 *   along with this program; if not, write to the                       *
 *   Free Software Foundation, Inc.,                                     *
 *   59 Temple Place - Suite 330, Boston, MA  02111-1307, USA.           *
 *************************************************************************/

#include "navier_mpi.h"
float nearbyintf(float x);


void Calcula_Espectro_Rodaja(void)
{
    int z,y,x,indice_z,indice_y,indice_x,indice_Matriz;
    int aux;
    FILE *DATOS;
//  Si hay que guardar en disco se comienza a fabricar el archivo
    if(GUARDAESTADO)
    {
        if(MPI.PROCESO_MPI==0)
        {
            if ((DATOS=fopen("espectro.html","w")) == NULL)
            {
                fprintf(stderr, "\nNo puedo abrir: espectro.html\n");
                exit(1);
            }

            fprintf(DATOS,"<html>\n<head><title>Espectro</title></head>\n<body><table>\
n<tr><td>Tiempo
total:</td><td>%.6e</td></tr>\n<tr><td>ETA:</td><td>%.6e</td></tr>\n<tr><td>K*ETA:</td>
<td>%.6e</td></tr>\n<tr><td>EPSILON:</td><td>%.6e</td></tr> </table>
\n\n<table>\n",TIEMPOTOTAL,ETA,ETA*K_FILTRO,EPSILON);
        }
    }
//  Se borran los valores anteriores de las variables para los cálculos
    Espectro.u_prima_cuadrado=0.0;
    Espectro.EPSILON=0.0;
    Espectro.L=0.0;
    Estadisticas.MaxEspectro=0.0;

    for(z=0;z<Espectro.tam;z++)
    {
        Espectro.Valores_Rodaja_MPI[z]=0.0;
    }

//  Se recorre el cubo en frecuencia sumando los valores en su componente espectral
adecuada
    for(y=0;y<TAM_LOCAL.NY;y++)
    {
        indice_y=((y+TAM_LOCAL.NY_offset)>=(N/2))?((y+TAM_LOCAL.NY_offset)-N):(y+TAM_LO
CAL.NY_offset);
        for(x=0;x<N;x++)
        {
            indice_x=(x>=(N/2))?(x-N):x;
            for(z=0;z<((N/2)+1);z++)
```



```
            {
                indice_z=(z>=(N/2))?(z-N):z;
                indice_Matriz=z+((N/2)+1)*(x+N*y);

                aux=(int)nearbyintf(sqrt(indice_z*indice_z+indice_x*indice_x+indice_y*i
ndice_y));
                if(aux<Espectro.tam)
                {
//                  Como solo tenemos la mitad del cubo las componentes fuera del
plano Z=0 cuentan dobles
                    if(indice_z==0)
                    {
    //                  Valor del espectro

                        Espectro.Valores_Rodaja_MPI[aux]+=0.5*(
                                        VX.Fred[indice_Matriz].re*VX.Fred[indice_Matriz].re
+VX.Fred[indice_Matriz].im*VX.Fred[indice_Matriz].im
                                        +
                                        VY.Fred[indice_Matriz].re*VY.Fred[indice_Matriz
].re+VY.Fred[indice_Matriz].im*VY.Fred[indice_Matriz].im
                                        +
                                        VZ.Fred[indice_Matriz].re*VZ.Fred[indice_Matriz
].re+VZ.Fred[indice_Matriz].im*VZ.Fred[indice_Matriz].im
                                        );
                    }
                    else
                    {
    //                  Valor del espectro
                        Espectro.Valores_Rodaja_MPI[aux]+=(
                                    VX.Fred[indice_Matriz].re*VX.Fred[indice_Matriz].re+VX.
Fred[indice_Matriz].im*VX.Fred[indice_Matriz].im
                                    +
                                    VY.Fred[indice_Matriz].re*VY.Fred[indice_Matriz].re
+VY.Fred[indice_Matriz].im*VY.Fred[indice_Matriz].im
                                    +
                                    VZ.Fred[indice_Matriz].re*VZ.Fred[indice_Matriz].re
+VZ.Fred[indice_Matriz].im*VZ.Fred[indice_Matriz].im
                                    );
                    }
                }
            }
        }
    }


//  Sumamos los datos de todos los procesadores
    MPI_Barrier(MPI_COMM_WORLD);
    MPI_Allreduce ( Espectro.Valores_Rodaja_MPI,Espectro.Valores_Rodaja, Espectro.tam,
MPI_DOUBLE,MPI_SUM, MPI_COMM_WORLD);

//  Calculamos las variables en frecuencia
    for(z=1;z<Espectro.tam;z++)
    {
//      Si se guarda el estado se escribe en disco
        if(GUARDAESTADO)
        {
            if(MPI.PROCESO_MPI==0)
            {
                fprintf(DATOS,"<tr> <td>%d</td> <td>%.6e</td>
</tr>\n",z,Espectro.Valores_Rodaja[z]);
            }
        }

//      Calculamos las integrales por el método del trapecio
        if (Estadisticas.MaxEspectro<Espectro.Valores_Rodaja[z])
            Estadisticas.MaxEspectro=Espectro.Valores_Rodaja[z];
        if (Estadisticas.MaxEspectro< (
pow(pow(z,5)/(EPSILON*EPSILON),1.0/3.0)*Espectro.Valores_Rodaja[z]))
```

```
            Estadisticas.MaxEspectro=(
pow(pow(z,5)/(EPSILON*EPSILON),1.0/3.0)*Espectro.Valores_Rodaja[ z] );

        Espectro.u_prima_cuadrado+=(Espectro.Valores_Rodaja[ z] +Espectro.Valores_Rodaja[
z-1] )*0.5;

        Espectro.EPSILON+=( (z*z*Espectro.Valores_Rodaja[ z] )
+((z-1)*(z-1)*Espectro.Valores_Rodaja[ z-1] ) ) *0.5;

        if(z>1)
        {
            Espectro.L+=( (Espectro.Valores_Rodaja[ z] /z)
+(Espectro.Valores_Rodaja[ z-1] /(z-1)) ) *0.5;
        }
        else
        {
            Espectro.L+=Espectro.Valores_Rodaja[ 1] ;
        }

    }

//  Terminamos los cálculos espectrales
    Espectro.u_prima_cuadrado*=2.0/3.0;
    Espectro.EPSILON*=2.0*VISCOSIDAD;
    Espectro.L*=M_PI_2/Espectro.u_prima_cuadrado;

    Espectro.Lambda=sqrt(15.0*VISCOSIDAD*Espectro.u_prima_cuadrado/Espectro.EPSILON);
    Espectro.Re_lambda=sqrt(Espectro.u_prima_cuadrado)*Espectro.Lambda/VISCOSIDAD;
    Espectro.T=Espectro.L/sqrt(Espectro.u_prima_cuadrado);

//  Si se guarda el estado se escribe en disco el final del archivo
    if(GUARDAESTADO)
    {
        if(MPI.PROCESO_MPI==0)
        {
            fprintf(DATOS,"\n</table>\n</body>\n</html>\n");
            fclose(DATOS);
        }
    }

//  Se genera la imagen  del espectro
    Genera_Imagen_Espectro_Rodaja();

    return;
}
```

```
/*************************************************************************
 *   Copyright (C) 2005 by Ignacio López Díaz                            *
 *   igna@us.es                                                          *
 *                                                                       *
 *   This program is free software; you can redistribute it and/or modify *
 *   it under the terms of the GNU General Public License as published by *
 *   the Free Software Foundation; either version 2 of the License, or   *
 *   (at your option) any later version.                                 *
 *                                                                       *
 *   This program is distributed in the hope that it will be useful,     *
 *   but WITHOUT ANY WARRANTY; without even the implied warranty of      *
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the       *
 *   GNU General Public License for more details.                        *
 *                                                                       *
 *   You should have received a copy of the GNU General Public License   *
 *   along with this program; if not, write to the                       *
 *   Free Software Foundation, Inc.,                                     *
 *   59 Temple Place - Suite 330, Boston, MA  02111-1307, USA.           *
 *************************************************************************/


#include "navier_mpi.h"

void Calcula_Invariantes(void)
{
    int x,z,y;

    FILE *DATOS;
    char nombrefichero[30];
    int indice_x,indice_z,indice_y,indice_Matriz;
    fftw_real divergencia,determinante,adjuntos;

//  Primero calculamos la divergencia

//  Borramos el valor máximo anterior
    Estadisticas.DIVMAX=0.0;

//  En el siguiente bucle calculamos el gradiente en frecuencia
    for(y=0;y<TAM_LOCAL.NY;y++)
    {
        indice_y=((y+TAM_LOCAL.NY_offset)>=(N/2))?((y+TAM_LOCAL.NY_offset)-N):(y+TAM_LOCAL.NY_offset);
        for(x=0;x<N;x++)
        {
            indice_x=(x>=(N/2))?(x-N):x;
            for(z=0;z<((N/2)+1);z++)
            {
                indice_z=(z>=(N/2))?(z-N):z;
                indice_Matriz=z+((N/2)+1)*(x+N*y);

//              Parte imaginaria
                Gradiente.XX.Fred[indice_Matriz].im=indice_x*VX.Fred[indice_Matriz].re;
                Gradiente.YY.Fred[indice_Matriz].im=indice_y*VY.Fred[indice_Matriz].re;
                Gradiente.ZZ.Fred[indice_Matriz].im=indice_z*VZ.Fred[indice_Matriz].re;
//              Parte real
                Gradiente.XX.Fred[indice_Matriz].re=-indice_x*VX.Fred[indice_Matriz].im;

                Gradiente.YY.Fred[indice_Matriz].re=-indice_y*VY.Fred[indice_Matriz].im;

                Gradiente.ZZ.Fred[indice_Matriz].re=-indice_z*VZ.Fred[indice_Matriz].im;
            }
        }
    }

//  Transformamos
    rfftwnd_mpi(PlanInv_mpi, 1, Gradiente.XX.Esp, EspacioAux.Esp,
FFTW_TRANSPOSED_ORDER);
```



```
    rfftwnd_mpi(PlanInv_mpi, 1, Gradiente.YY.Esp, EspacioAux.Esp,
FFTW_TRANSPOSED_ORDER);
    rfftwnd_mpi(PlanInv_mpi, 1, Gradiente.ZZ.Esp, EspacioAux.Esp,
FFTW_TRANSPOSED_ORDER);

    for(x=0;x<TAM_LOCAL.NX;x++)
    {
        for(y=0;y<N;y++)
        {
            for(z=0;z<N;z++)
            {
                indice_Matriz=z+(2*((N/2)+1))*(y+N*x);

                divergencia=Gradiente.XX.Esp[indice_Matriz]+Gradiente.YY.Esp[indice_Matriz]+Gradiente.ZZ.Esp[indice_Matriz];

                if(fabs(Estadisticas.DIVMAX)<fabs(divergencia))
                    Estadisticas.DIVMAX=divergencia;
            }
        }

    }

//  Si no se calculan el resto de los invariantes salimos
    if(CALCULAINVARIANTES==0)return;

//  Ahora calculamos el resto de invariantes
//  En el siguiente bucle calculamos el gradiente en frecuencia
    for(y=0;y<TAM_LOCAL.NY;y++)
    {
        indice_y=((y+TAM_LOCAL.NY_offset)>=(N/2))?((y+TAM_LOCAL.NY_offset)-N):(y+TAM_LOCAL.NY_offset);
        for(x=0;x<N;x++)
        {
            indice_x=(x>=(N/2))?(x-N):x;
            for(z=0;z<((N/2)+1);z++)
            {
                indice_z=(z>=(N/2))?(z-N):z;
                indice_Matriz=z+((N/2)+1)*(x+N*y);

//              Parte imaginaria
                Gradiente.YX.Fred[indice_Matriz].im=indice_x*VY.Fred[indice_Matriz].re;
                Gradiente.ZX.Fred[indice_Matriz].im=indice_x*VZ.Fred[indice_Matriz].re;
                Gradiente.XY.Fred[indice_Matriz].im=indice_y*VX.Fred[indice_Matriz].re;
                Gradiente.ZY.Fred[indice_Matriz].im=indice_y*VZ.Fred[indice_Matriz].re;
                Gradiente.XZ.Fred[indice_Matriz].im=indice_z*VX.Fred[indice_Matriz].re;
                Gradiente.YZ.Fred[indice_Matriz].im=indice_z*VY.Fred[indice_Matriz].re;

//              Parte real
                Gradiente.YX.Fred[indice_Matriz].re=-indice_x*VY.Fred[indice_Matriz].im;

                Gradiente.ZX.Fred[indice_Matriz].re=-indice_x*VZ.Fred[indice_Matriz].im;

                Gradiente.XY.Fred[indice_Matriz].re=-indice_y*VX.Fred[indice_Matriz].im;

                Gradiente.ZY.Fred[indice_Matriz].re=-indice_y*VZ.Fred[indice_Matriz].im;

                Gradiente.XZ.Fred[indice_Matriz].re=-indice_z*VX.Fred[indice_Matriz].im;

                Gradiente.YZ.Fred[indice_Matriz].re=-indice_z*VY.Fred[indice_Matriz].im;
            }
        }
    }

//  Transformamos

    rfftwnd_mpi(PlanInv_mpi, 1, Gradiente.XY.Esp, EspacioAux.Esp,
```

```
FFTW_TRANSPOSED_ORDER);
    rfftwnd_mpi(PlanInv_mpi, 1, Gradiente.XZ.Esp, EspacioAux.Esp,
FFTW_TRANSPOSED_ORDER);

    rfftwnd_mpi(PlanInv_mpi, 1, Gradiente.YX.Esp, EspacioAux.Esp,
FFTW_TRANSPOSED_ORDER);
    rfftwnd_mpi(PlanInv_mpi, 1, Gradiente.YZ.Esp, EspacioAux.Esp,
FFTW_TRANSPOSED_ORDER);

    rfftwnd_mpi(PlanInv_mpi, 1, Gradiente.ZX.Esp, EspacioAux.Esp,
FFTW_TRANSPOSED_ORDER);
    rfftwnd_mpi(PlanInv_mpi, 1, Gradiente.ZY.Esp, EspacioAux.Esp,
FFTW_TRANSPOSED_ORDER);

    Inicializa_Imagen_Invariantes();

//  Escribimos los datos de los invariantes

    MPI_Barrier(MPI_COMM_WORLD);
//  Si se guarda el estado escribimos los invariantes en archivos temporales locales
por velocidad que luego se recomponen en uno
    if(GUARDAESTADO)
    {
        sprintf(nombrefichero,"temporal/invariantes-P%d.dat",MPI.PROCESO_MPI);
        if ((DATOS=fopen(nombrefichero,"wb")) == NULL)
        {
            fprintf(stderr, "\nNo puedo abrir invariantes.dat\n");
            exit(1);
        }
    }

//  Recorremos el cubo generando los invariantes
    for(x=0;x<TAM_LOCAL.NX;x++)
    {
        for(y=0;y<N;y++)
        {
            for(z=0;z<N;z++)
            {
                indice_Matriz=z+(2*((N/2)+1))*(y+N*x);

                adjuntos=-
                        (Gradiente.XY.Esp[ indice_Matriz]*Gradiente.YX.Esp[ indice_Matriz
])-
                        (Gradiente.XZ.Esp[ indice_Matriz]*Gradiente.ZX.Esp[ indice_Matriz
])-
                        (Gradiente.ZY.Esp[ indice_Matriz]*Gradiente.YZ.Esp[ indice_Matriz
])
                        -0.5*(
                        (Gradiente.XX.Esp[ indice_Matriz]*Gradiente.XX.Esp[ indice_Matriz
])
                        +
                        (Gradiente.YY.Esp[ indice_Matriz]*Gradiente.YY.Esp[ indice_Matriz
])
                        +
                        (Gradiente.ZZ.Esp[ indice_Matriz]*Gradiente.ZZ.Esp[ indice_Matriz
])
                            )
                        ;

                determinante=(-1.0/3.0)*(
                        (Gradiente.XX.Esp[ indice_Matriz]*Gradiente.XX.Esp[ indice_Matriz
]*Gradiente.XX.Esp[ indice_Matriz])+
                        (Gradiente.XX.Esp[ indice_Matriz]*Gradiente.XY.Esp[ indice_Matriz
]*Gradiente.YX.Esp[ indice_Matriz])+
                        (Gradiente.XX.Esp[ indice_Matriz]*Gradiente.XZ.Esp[ indice_Matriz
]*Gradiente.ZX.Esp[ indice_Matriz])+

                        (Gradiente.XY.Esp[ indice_Matriz]*Gradiente.YX.Esp[ indice_Matriz
```



```
]*Gradiente.XX.Esp[ indice_Matriz])+
                        (Gradiente.XY.Esp[ indice_Matriz]*Gradiente.YY.Esp[ indice_Matriz
]*Gradiente.YX.Esp[ indice_Matriz])+
                        (Gradiente.XY.Esp[ indice_Matriz]*Gradiente.YZ.Esp[ indice_Matriz
]*Gradiente.ZX.Esp[ indice_Matriz])+

                        (Gradiente.XZ.Esp[ indice_Matriz]*Gradiente.ZX.Esp[ indice_Matriz
]*Gradiente.XX.Esp[ indice_Matriz])+
                        (Gradiente.XZ.Esp[ indice_Matriz]*Gradiente.ZY.Esp[ indice_Matriz
]*Gradiente.YX.Esp[ indice_Matriz])+
                        (Gradiente.XZ.Esp[ indice_Matriz]*Gradiente.ZZ.Esp[ indice_Matriz
]*Gradiente.ZX.Esp[ indice_Matriz])+

                        (Gradiente.YX.Esp[ indice_Matriz]*Gradiente.XX.Esp[ indice_Matriz
]*Gradiente.XY.Esp[ indice_Matriz])+
                        (Gradiente.YX.Esp[ indice_Matriz]*Gradiente.XY.Esp[ indice_Matriz
]*Gradiente.YY.Esp[ indice_Matriz])+
                        (Gradiente.YX.Esp[ indice_Matriz]*Gradiente.XZ.Esp[ indice_Matriz
]*Gradiente.ZY.Esp[ indice_Matriz])+

                        (Gradiente.YY.Esp[ indice_Matriz]*Gradiente.YX.Esp[ indice_Matriz
]*Gradiente.XY.Esp[ indice_Matriz])+
                        (Gradiente.YY.Esp[ indice_Matriz]*Gradiente.YY.Esp[ indice_Matriz
]*Gradiente.YY.Esp[ indice_Matriz])+
                        (Gradiente.YY.Esp[ indice_Matriz]*Gradiente.YZ.Esp[ indice_Matriz
]*Gradiente.ZY.Esp[ indice_Matriz])+

                        (Gradiente.YZ.Esp[ indice_Matriz]*Gradiente.ZX.Esp[ indice_Matriz
]*Gradiente.XY.Esp[ indice_Matriz])+
                        (Gradiente.YZ.Esp[ indice_Matriz]*Gradiente.ZY.Esp[ indice_Matriz
]*Gradiente.YY.Esp[ indice_Matriz])+
                        (Gradiente.YZ.Esp[ indice_Matriz]*Gradiente.ZZ.Esp[ indice_Matriz
]*Gradiente.ZY.Esp[ indice_Matriz])+

                        (Gradiente.ZX.Esp[ indice_Matriz]*Gradiente.XX.Esp[ indice_Matriz
]*Gradiente.XZ.Esp[ indice_Matriz])+
                        (Gradiente.ZX.Esp[ indice_Matriz]*Gradiente.XY.Esp[ indice_Matriz
]*Gradiente.YZ.Esp[ indice_Matriz])+
                        (Gradiente.ZX.Esp[ indice_Matriz]*Gradiente.XZ.Esp[ indice_Matriz
]*Gradiente.ZZ.Esp[ indice_Matriz])+

                        (Gradiente.ZY.Esp[ indice_Matriz]*Gradiente.YX.Esp[ indice_Matriz
]*Gradiente.XZ.Esp[ indice_Matriz])+
                        (Gradiente.ZY.Esp[ indice_Matriz]*Gradiente.YY.Esp[ indice_Matriz
]*Gradiente.YZ.Esp[ indice_Matriz])+
                        (Gradiente.ZY.Esp[ indice_Matriz]*Gradiente.YZ.Esp[ indice_Matriz
]*Gradiente.ZZ.Esp[ indice_Matriz])+

                        (Gradiente.ZZ.Esp[ indice_Matriz]*Gradiente.ZX.Esp[ indice_Matriz
]*Gradiente.XZ.Esp[ indice_Matriz])+
                        (Gradiente.ZZ.Esp[ indice_Matriz]*Gradiente.ZY.Esp[ indice_Matriz
]*Gradiente.YZ.Esp[ indice_Matriz])+
                        (Gradiente.ZZ.Esp[ indice_Matriz]*Gradiente.ZZ.Esp[ indice_Matriz
]*Gradiente.ZZ.Esp[ indice_Matriz])
                                    );


                determinante*=pow((ETA/Estadisticas.u_Media_XX),1.5);
                adjuntos*=(ETA/Estadisticas.u_Media_XX);

//              Ponemos el  punto en la imagen
                Poner_Punto_Imagen_Invariantes(determinante,adjuntos);

//              Si se guarda el estado se escribe en disco
                if(GUARDAESTADO)
                {
                    fwrite(&divergencia,sizeof(fftw_real),1,DATOS);
                    fwrite(&determinante,sizeof(fftw_real),1,DATOS);
```

```
                    fwrite(&adjuntos,sizeof(fftw_real),1,DATOS);
                }
            }
        }

    }

//  Cerramos el archivo si es necesario
    if(GUARDAESTADO)fclose(DATOS);

//  Generamos la imagen de invariantes
    Genera_Imagen_Invariantes();

    return;
}
```

```
/*************************************************************************
 *   Copyright (C) 2005 by Ignacio López Díaz                            *
 *   igna@us.es                                                          *
 *                                                                       *
 *   This program is free software; you can redistribute it and/or modify *
 *   it under the terms of the GNU General Public License as published by *
 *   the Free Software Foundation; either version 2 of the License, or   *
 *   (at your option) any later version.                                 *
 *                                                                       *
 *   This program is distributed in the hope that it will be useful,     *
 *   but WITHOUT ANY WARRANTY; without even the implied warranty of      *
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the       *
 *   GNU General Public License for more details.                        *
 *                                                                       *
 *   You should have received a copy of the GNU General Public License   *
 *   along with this program; if not, write to the                       *
 *   Free Software Foundation, Inc.,                                     *
 *   59 Temple Place - Suite 330, Boston, MA  02111-1307, USA.           *
 *************************************************************************/


#include "navier_mpi.h"


void Escribe_Estadisticas(void)
{
//  En esta función se sacan los valores instantáneos por pantalla
    if(MPI.PROCESO_MPI==0)
    {
        setlocale(LC_ALL,"");

        printf(BORRAR_PANTALLA);
        printf("Navier-Stokes  (%dx%dx%d)\n",N,N,N);
        printf("Tipo de integración: %s.\n",TIPOINT);
        printf("Tiempo de integración: %+.6e\tPaso de integración
%e\n",TIEMPOTOTAL,dt);
        printf("Viscosidad= %+.6e\tViscosidad negativa=
%+.6e\n",VISCOSIDAD,VISCOSIDADNEG);
        printf("<u_x>= %+.6e\t<u_y>= %+.6e\t<u_z>=
%+.6e\n",Estadisticas.u_Media_X,Estadisticas.u_Media_Y,Estadisticas.u_Media_Z);
        printf("u'_x= %+.6e\tu'_y= %+.6e\tu'_z=
%+.6e\n",Estadisticas.u_Media_XX,Estadisticas.u_Media_YY,Estadisticas.u_Media_ZZ);
        printf("<u_x*u_y>= %+.6e\t<u_y*u_z>= %+.6e\t<u_z*u_x>=
%+.6e",Estadisticas.u_Media_XY,Estadisticas.u_Media_YZ,Estadisticas.u_Media_ZX);
        printf("\n%sEPSILON= %s%+.6e%s\tETA= %+.6e\tK*ETA=
%s%+.6e%s\n",AZUL,MAGENTA,EPSILON,AZUL, ETA, ROJO ,Estadisticas.KETA,NADA);
        printf("%sEPSILON_espectro= %s%+.6e%s\tK*ETA_espectro=
%s%+.6e%s",AZUL,MAGENTA,Espectro.EPSILON,AZUL,ROJO,
        Estadisticas.KETA_espectro,NADA);

        printf("\nu'²_espectro=
%+.6e\tu'²=%e",Espectro.u_prima_cuadrado,(Estadisticas.u_Media_XX*Estadisticas.u_Media_
XX+Estadisticas.u_Media_YY*Estadisticas.u_Media_YY+Estadisticas.u_Media_ZZ*Estadisticas
.u_Media_ZZ)/3.0);

        printf("\nL_espectro= %+.6e\tLambda_espectro=
%+.6e",Espectro.L,Espectro.Lambda);
        printf("\nRe_lambda_espectro= %+.6e\tT_espectro=
%+.6e\n",Espectro.Re_lambda,Espectro.T);
        printf("\nv_k= %+.6e\tv_k_espectro=
%+.6e\n",pow(EPSILON,0.25),pow(Espectro.EPSILON,0.25));

        printf("\nDIVMAX= %+.6e\tNúmero de eliminaciones de la divergencia:
%d\n",Estadisticas.DIVMAX,Estadisticas.NumCambiosDIVMAX);

        #if (TIPO_KETA==KETA_VORTICIDAD)
    //      Calculo con el EPSILON a partir de la vorticidad
            printf("\nEscala de tiempo de Kolmogorov: %st_eta= %s%+.6e%s\tdt/t_eta=
```



```
%s%+.6f%s",AZUL,FONDO_ROJO,sqrt(VISCOSIDAD/EPSILON),AZUL,FONDO_ROJO,dt/sqrt(VISCOSIDAD/
EPSILON),NADA);
            printf("\nT_e/t_eta= %+.6e",Espectro.T/sqrt(VISCOSIDAD/Espectro.EPSILON));
        #elif (TIPO_KETA==KETA_ESPECTRO)
    //      Calculo con el EPSILON a partir del espectro
            printf("\nEscala de tiempo de Kolmogorov: %st_eta= %s%+.6e%s\tdt/t_eta=
%s%+.6f%s",AZUL,FONDO_ROJO,sqrt(VISCOSIDAD/Espectro.EPSILON),AZUL,FONDO_ROJO,dt/sqrt(VI
SCOSIDAD/Espectro.EPSILON),NADA);
            printf("\nT_e/t_eta= %+.6e",Espectro.T/sqrt(VISCOSIDAD/Espectro.EPSILON));

        #endif

        printf("\n%sK_FILTRO= %s%+.2f%s /
%s%+.2f%s",AZUL,FONDO_ROJO,K_FILTRO,AZUL,FONDO_ROJO,N/2.0-1.5,NADA);
        printf("\n%sK^2_FILTRO= %s%+.2f%s /
%s%+.2f%s",AZUL,FONDO_ROJO,Kcuadrado_FILTRO,AZUL,FONDO_ROJO,(N/2.0-1.5)*(N/2.0-1.5),NAD
A);


        printf("\n\n%sL= %+.6e",AZUL,Espectro.L);
        printf("\nL/Lambda= %+.6e",Espectro.L/Espectro.Lambda);
        printf("\nL/ETA= %+.6e",Espectro.L/ETA);
        printf("\nEPSILON*L/u'³=
%+.6e",Espectro.EPSILON*Espectro.L/pow(Espectro.u_prima_cuadrado,3.0/2.0));
        printf("\nomega_prima*T= %+.6e",Estadisticas.omega_prima*Espectro.T);
        printf("\nt/T= %+.6e%s",TIEMPOTOTAL/Espectro.T,NADA);

        printf("\nPID.K_P= %+.6e\tPID.K_I= %+.6e\tPID.K_D=
%+.6e",PID.K_P,PID.K_I,PID.K_D);
        printf("\nPID.error= %+.6e\tPID.int_error= %+.6e\tPID.deriv_error=
%+.6e",PID.error,PID.int_error,PID.deriv_error);

        printf("\nPID.indice_KETA= %d",PID.indice_KETA);
        printf("\tCuantos= %d\n",Estadisticas.cuantos);
        fflush(stdout);
    }
    MPI_Barrier(MPI_COMM_WORLD);

//  Si ETA está fuera de los límites finalizamos con un mensaje de error
    if(isnan(ETA)||(ETA*N<1e-1))
    {
        if(MPI.PROCESO_MPI==0)printf("\nError: eta fuera de los límites.\n");
        fflush(stdout);
        Finalizacion();
        MPI_Finalize();
        exit(1);

    }
    return;
}


void Calcula_VISCOSIDADNEG(void)
{
//  En esta función se calcula la viscosidad negativa necesaria para los cálculos
integrales.

//  Primero se calcula KETA por los dos métodos
    Estadisticas.KETA=ETA*K_FILTRO;
    Estadisticas.KETA_espectro=pow(VISCOSIDAD*VISCOSIDAD*VISCOSIDAD/Espectro.EPSILON,0.
25)*K_FILTRO;

//  Calculamos el error
    PID.error_ant=PID.error;

    #if (TIPO_KETA==KETA_VORTICIDAD)
//      Cálculos con el EPSILON a partir de la vorticidad
        PID.error=KETA_ESTACIONARIA-Estadisticas.KETA;
```

```
    #elif (TIPO_KETA==KETA_ESPECTRO)
//      Cálculos con el EPSILON a partir del espectro
        PID.error=KETA_ESTACIONARIA-Estadisticas.KETA_espectro;
    #endif


//  Calculamos la derivada y la integral del error
    PID.deriv_error=(PID.error-PID.error_ant)/dt;
    PID.int_error+=PID.error*dt;

//  Si se usa el forzado se calcula el PID si no se pone la viscosidad normal
    if(FORZADO)
    {
    //  Calculamos el PID- No se usa usaremos un controlador PI en su lugar
    //
VISCOSIDADNEG=(145.0*PID.error+50.0*PID.int_error+5.0*PID.deriv_error)*VISCOSIDAD;

    //  Controlador PI
        VISCOSIDADNEG=(PID.K_P*PID.error+PID.K_I*PID.int_error+PID.K_D*PID.deriv_error)
*VISCOSIDAD;
    }
    else
    {
        VISCOSIDADNEG=VISCOSIDAD;
    }

//  Finalmente se genera una imagen de la evolución de KETA
    Genera_Imagen_KETA(Estadisticas.KETA,Estadisticas.KETA_espectro);

    return;
}


void RecuperaEstado(char *Nombre,int cabecera)
{
//  Función que lee el archivo de estado
    MPI_File DATOS_MPI;
    int offset;
//  Usamos el sistema de archivos con MPI
    MPI_Barrier(MPI_COMM_WORLD);

//  Abrimos el fichero entre todos los nodos
    if(MPI_File_open (MPI_COMM_WORLD, Nombre,  MPI_MODE_RDONLY, MPI_INFO_NULL,
&DATOS_MPI)!=MPI_SUCCESS)
    {
        if(MPI.PROCESO_MPI==0)
        {
            fprintf(stderr, "\nNo puedo abrir %s\n",Nombre);
        }
        exit(1);
    }

//  La cabecera la leen todos  a la vez
    MPI_File_read_all(DATOS_MPI, &N,sizeof(unsigned int), MPI_CHARACTER, &MPI.Estado);
    MPI_File_read_all(DATOS_MPI, &N3D,sizeof(unsigned int), MPI_CHARACTER,
&MPI.Estado);
    MPI_File_read_all(DATOS_MPI, &N3D_DIV,sizeof(unsigned int), MPI_CHARACTER,
&MPI.Estado);
    MPI_File_read_all(DATOS_MPI, &N3D_FREC,sizeof(unsigned int), MPI_CHARACTER,
&MPI.Estado);

    MPI_File_read_all(DATOS_MPI, &K_FILTRO,sizeof(fftw_real), MPI_CHARACTER,
&MPI.Estado);
    MPI_File_read_all(DATOS_MPI, &Kcuadrado_FILTRO,sizeof(fftw_real), MPI_CHARACTER,
&MPI.Estado);
    MPI_File_read_all(DATOS_MPI, &K_VISCOSIDAD,sizeof(fftw_real), MPI_CHARACTER,
&MPI.Estado);
    MPI_File_read_all(DATOS_MPI, &K_VISCOSIDAD_CUAD,sizeof(fftw_real), MPI_CHARACTER,
```



```
&MPI.Estado);

    MPI_File_read_all(DATOS_MPI, &VISCOSIDAD,sizeof(fftw_real), MPI_CHARACTER,
&MPI.Estado);
    MPI_File_read_all(DATOS_MPI, &VISCOSIDADNEG,sizeof(fftw_real), MPI_CHARACTER,
&MPI.Estado);
    MPI_File_read_all(DATOS_MPI, &EPSILON,sizeof(fftw_real), MPI_CHARACTER,
&MPI.Estado);
    MPI_File_read_all(DATOS_MPI, &ETA,sizeof(fftw_real), MPI_CHARACTER, &MPI.Estado);
    MPI_File_read_all(DATOS_MPI, &KETA_ESTACIONARIA,sizeof(fftw_real), MPI_CHARACTER,
&MPI.Estado);
    MPI_File_read_all(DATOS_MPI, &TIEMPOTOTAL,sizeof(fftw_real), MPI_CHARACTER,
&MPI.Estado);
    MPI_File_read_all(DATOS_MPI, &dt,sizeof(fftw_real), MPI_CHARACTER, &MPI.Estado);

    MPI_File_read_all(DATOS_MPI, &FORZADO,sizeof(int), MPI_CHARACTER, &MPI.Estado);
    MPI_File_read_all(DATOS_MPI, &CALCULAINVARIANTES,sizeof(int), MPI_CHARACTER,
&MPI.Estado);

    MPI_File_read_all(DATOS_MPI, &PID.error,sizeof(fftw_real), MPI_CHARACTER,
&MPI.Estado);
    MPI_File_read_all(DATOS_MPI, &PID.deriv_error,sizeof(fftw_real), MPI_CHARACTER,
&MPI.Estado);
    MPI_File_read_all(DATOS_MPI, &PID.int_error,sizeof(fftw_real), MPI_CHARACTER,
&MPI.Estado);
    MPI_File_read_all(DATOS_MPI, &PID.error_ant,sizeof(fftw_real), MPI_CHARACTER,
&MPI.Estado);
    MPI_File_read_all(DATOS_MPI, &PID.K_P,sizeof(fftw_real), MPI_CHARACTER,
&MPI.Estado);
    MPI_File_read_all(DATOS_MPI, &PID.K_I,sizeof(fftw_real), MPI_CHARACTER,
&MPI.Estado);
    MPI_File_read_all(DATOS_MPI, &PID.K_D,sizeof(fftw_real), MPI_CHARACTER,
&MPI.Estado);

    MPI_File_read_all(DATOS_MPI, &Espectro.u_prima_cuadrado,sizeof(fftw_real),
MPI_CHARACTER, &MPI.Estado);
    MPI_File_read_all(DATOS_MPI, &Espectro.EPSILON,sizeof(fftw_real), MPI_CHARACTER,
&MPI.Estado);
    MPI_File_read_all(DATOS_MPI, &Espectro.L,sizeof(fftw_real), MPI_CHARACTER,
&MPI.Estado);

    MPI_File_read_all(DATOS_MPI, &Imagen.alto,sizeof(int), MPI_CHARACTER, &MPI.Estado);
    MPI_File_read_all(DATOS_MPI, &Imagen.ancho,sizeof(int), MPI_CHARACTER,
&MPI.Estado);
    MPI_File_read_all(DATOS_MPI, &Imagen.Divisiones_X,sizeof(int), MPI_CHARACTER,
&MPI.Estado);
    MPI_File_read_all(DATOS_MPI, &Imagen.Divisiones_Y,sizeof(int), MPI_CHARACTER,
&MPI.Estado);
    MPI_File_read_all(DATOS_MPI, &Imagen.MARGEN_X,sizeof(int), MPI_CHARACTER,
&MPI.Estado);
    MPI_File_read_all(DATOS_MPI, &Imagen.MARGEN_Y,sizeof(int), MPI_CHARACTER,
&MPI.Estado);
    MPI_File_read_all(DATOS_MPI, &Imagen.Puntos_Por_Division_X,sizeof(int),
MPI_CHARACTER, &MPI.Estado);
    MPI_File_read_all(DATOS_MPI, &Imagen.Puntos_Por_Division_Y,sizeof(int),
MPI_CHARACTER, &MPI.Estado);
    MPI_File_read_all(DATOS_MPI, &Imagen.KETA_MAX,sizeof(int), MPI_CHARACTER,
&MPI.Estado);
    MPI_File_read_all(DATOS_MPI, &Imagen.KETA_MIN,sizeof(int), MPI_CHARACTER,
&MPI.Estado);

//  Calculamos los datos redundantes
    Imagen.Divisiones_X=2+floor(log10(N/2));
    Imagen.Puntos_Por_Division_X=
(Imagen.ancho-2*Imagen.MARGEN_X)/(Imagen.Divisiones_X-1);
    Imagen.Puntos_Por_Division_Y=(Imagen.alto-2*Imagen.MARGEN_Y)/(Imagen.Divisiones_Y);

//  Si hay que leer el resto de los datos se hace leyendo cada proceso su parte
```

```
    if(cabecera==0)
    {
        //  Leemos el estado

        offset=14*sizeof(int)+23*sizeof(fftw_real);
        MPI_File_read_at(DATOS_MPI,offset+
(TAM_LOCAL.NY_offset*(N*(N/2+1)))*sizeof(fftw_complex),VX.Frec,(TAM_LOCAL.NY*(N*(N/2+1)
))*sizeof(fftw_complex), MPI_CHARACTER, &MPI.Estado);

        offset=14*sizeof(int)+23*sizeof(fftw_real)+N3D*sizeof(fftw_real);
        MPI_File_read_at(DATOS_MPI,offset+
(TAM_LOCAL.NY_offset*(N*(N/2+1)))*sizeof(fftw_complex),VY.Frec,(TAM_LOCAL.NY*(N*(N/2+1)
))*sizeof(fftw_complex), MPI_CHARACTER, &MPI.Estado);

        offset=14*sizeof(int)+23*sizeof(fftw_real)+2*N3D*sizeof(fftw_real);
        MPI_File_read_at(DATOS_MPI,offset+
(TAM_LOCAL.NY_offset*(N*(N/2+1)))*sizeof(fftw_complex),VZ.Frec,(TAM_LOCAL.NY*(N*(N/2+1)
))*sizeof(fftw_complex), MPI_CHARACTER, &MPI.Estado);

//      Leemos VVORT

        offset=14*sizeof(int)+23*sizeof(fftw_real)+3*N3D*sizeof(fftw_real);
        MPI_File_read_at(DATOS_MPI,offset+
(TAM_LOCAL.NY_offset*(N*(N/2+1)))*sizeof(fftw_complex),VVORTX.Frec,(TAM_LOCAL.NY*(N*(N/
2+1)))*sizeof(fftw_complex), MPI_CHARACTER, &MPI.Estado);

        offset=14*sizeof(int)+23*sizeof(fftw_real)+4*N3D*sizeof(fftw_real);
        MPI_File_read_at(DATOS_MPI,offset+
(TAM_LOCAL.NY_offset*(N*(N/2+1)))*sizeof(fftw_complex),VVORTY.Frec,(TAM_LOCAL.NY*(N*(N/
2+1)))*sizeof(fftw_complex), MPI_CHARACTER, &MPI.Estado);

        offset=14*sizeof(int)+23*sizeof(fftw_real)+5*N3D*sizeof(fftw_real);
        MPI_File_read_at(DATOS_MPI,offset+
(TAM_LOCAL.NY_offset*(N*(N/2+1)))*sizeof(fftw_complex),VVORTZ.Frec,(TAM_LOCAL.NY*(N*(N/
2+1)))*sizeof(fftw_complex), MPI_CHARACTER, &MPI.Estado);

//      Leemos el estado nuevo
        offset=14*sizeof(int)+23*sizeof(fftw_real)+6*N3D*sizeof(fftw_real);
        MPI_File_read_at(DATOS_MPI,offset+
(TAM_LOCAL.NY_offset*(N*(N/2+1)))*sizeof(fftw_complex),VX_nuevo.Frec,(TAM_LOCAL.NY*(N*(
N/2+1)))*sizeof(fftw_complex), MPI_CHARACTER, &MPI.Estado);

        offset=14*sizeof(int)+23*sizeof(fftw_real)+7*N3D*sizeof(fftw_real);
        MPI_File_read_at(DATOS_MPI,offset+
(TAM_LOCAL.NY_offset*(N*(N/2+1)))*sizeof(fftw_complex),VY_nuevo.Frec,(TAM_LOCAL.NY*(N*(
N/2+1)))*sizeof(fftw_complex), MPI_CHARACTER, &MPI.Estado);

        offset=14*sizeof(int)+23*sizeof(fftw_real)+8*N3D*sizeof(fftw_real);
        MPI_File_read_at(DATOS_MPI,offset+
(TAM_LOCAL.NY_offset*(N*(N/2+1)))*sizeof(fftw_complex),VZ_nuevo.Frec,(TAM_LOCAL.NY*(N*(
N/2+1)))*sizeof(fftw_complex), MPI_CHARACTER, &MPI.Estado);

//      Leemos VVORT anterior
        offset=14*sizeof(int)+23*sizeof(fftw_real)+9*N3D*sizeof(fftw_real);
        MPI_File_read_at(DATOS_MPI,offset+
(TAM_LOCAL.NY_offset*(N*(N/2+1)))*sizeof(fftw_complex),VVORTX_ant.Frec,(TAM_LOCAL.NY*(N
*(N/2+1)))*sizeof(fftw_complex), MPI_CHARACTER, &MPI.Estado);

        offset=14*sizeof(int)+23*sizeof(fftw_real)+10*N3D*sizeof(fftw_real);
        MPI_File_read_at(DATOS_MPI,offset+
(TAM_LOCAL.NY_offset*(N*(N/2+1)))*sizeof(fftw_complex),VVORTY_ant.Frec,(TAM_LOCAL.NY*(N
*(N/2+1)))*sizeof(fftw_complex), MPI_CHARACTER, &MPI.Estado);

        offset=14*sizeof(int)+23*sizeof(fftw_real)+11*N3D*sizeof(fftw_real);
        MPI_File_read_at(DATOS_MPI,offset+
(TAM_LOCAL.NY_offset*(N*(N/2+1)))*sizeof(fftw_complex),VVORTZ_ant.Frec,(TAM_LOCAL.NY*(N
*(N/2+1)))*sizeof(fftw_complex), MPI_CHARACTER, &MPI.Estado);
```



```
    }

//  Cerramos el fichero y salimos
    MPI_File_close(&DATOS_MPI);
    return;
}




void LeeParametros (void)
{
//  Esta función lee los parámetros de entrada del fichero y los modifica
    char buffer[40];
    int n,m,modifica_imagen=0;
    FILE *DATOS;
    setlocale(LC_ALL,"es_ES@euro");
    if ((DATOS=fopen("Parametros.dat","r")) != NULL)
    {
//      Lee línea a línea y realiza las operaciones adecuadas
        while(fgets(buffer,40,DATOS)!=NULL)
        {
            if( strncmp(buffer, (const char *)&"FORZADO:", 8) ==0 )
            {
                FORZADO=atoi(&buffer[8]);
            }
            if( strncmp(buffer, (const char *)&"GUARDAESTADO:", 8) ==0 )
            {
                GUARDAESTADO=atoi(&buffer[13]);
            }
            if( strncmp(buffer, (const char *)&"CALCULAINVARIANTES:", 19) ==0 )
            {
                CALCULAINVARIANTES=atoi(&buffer[19]);
            }
            if( strncmp(buffer, (const char *)&"Pasos:", 6) ==0 )
            {
                Pasos=atoi(&buffer[6]);
            }
            if( strncmp(buffer, (const char *)&"VISCOSIDAD:", 11) ==0 )
            {
                VISCOSIDAD=atof(&buffer[11]);
            }
            if( strncmp(buffer, (const char *)&"dt:", 3) ==0 )
            {
                if(fabs(dt-atof(&buffer[3]))>1e-10)
                {
                    dt_nuevo=atof(&buffer[3]);
                    if(CAMBIAR_dt>=0)CAMBIAR_dt=1;
                }
            }
            else if( strncmp(buffer,(const char *) &"K_FILTRO:", 9) ==0 )
            {
                K_FILTRO=atof(&buffer[9]);
                Kcuadrado_FILTRO=(K_FILTRO*K_FILTRO);
                ETA_ESTACIONARIA=KETA_ESTACIONARIA/K_FILTRO;
                EPSILON_ESTACIONARIO=VISCOSIDAD*VISCOSIDAD*VISCOSIDAD/pow(ETA_ESTACIONA
RIA,4.0);
            }
            else if( strncmp(buffer,(const char *) &"CURSOR_K:", 9) ==0 )
            {
                CURSOR_K=atof(&buffer[9]);
            }
            else if( strncmp(buffer,(const char *) &"CURSOR_E:", 9) ==0 )
            {
                CURSOR_E=atof(&buffer[9]);
            }
```

```
            else if( strncmp(buffer,(const char *) &"KETA_ESTACIONARIA:", 18) ==0 )
            {
                KETA_ESTACIONARIA=atof(&buffer[ 18]);
                ETA_ESTACIONARIA=KETA_ESTACIONARIA/K_FILTRO;
                EPSILON_ESTACIONARIO=VISCOSIDAD*VISCOSIDAD*VISCOSIDAD/pow(ETA_ESTACIONA
RIA,4.0);
            }
            else if( strncmp(buffer,(const char *) &"TIEMPOTOTAL:", 12) ==0 )
            {
                TIEMPOTOTAL=atof(&buffer[ 12]);
            }
            else if( strncmp(buffer,(const char *) &"PID_K_P:", 8) ==0 )
            {
                PID.K_P=atof(&buffer[ 8]);
            }
            else if( strncmp(buffer,(const char *) &"PID_K_I:", 8) ==0 )
            {
                PID.K_I=atof(&buffer[ 8]);
            }
            else if( strncmp(buffer,(const char *) &"PID_K_D:", 8) ==0 )
            {
                PID.K_D=atof(&buffer[ 8]);
            }
            else if( strncmp(buffer,(const char *) &"PID_error:", 10) ==0 )
            {
                PID.error=atof(&buffer[ 10]);
            }
            else if( strncmp(buffer,(const char *) &"PID_deriv_error:", 16) ==0 )
            {
                PID.deriv_error=atof(&buffer[ 16]);
            }
            else if( strncmp(buffer,(const char *) &"PID_int_error:", 14) ==0 )
            {
                PID.int_error=atof(&buffer[ 14]);
            }

            else if( strncmp(buffer,(const char *) &"Imagen_Ancho:", 13) ==0 )
            {
                Imagen.ancho=atoi(&buffer[ 13]);
                modifica_imagen=1;
            }
            else if( strncmp(buffer,(const char *) &"Imagen_Alto:", 12) ==0 )
            {
                Imagen.alto=atoi(&buffer[ 12]);
                modifica_imagen=1;
            }
            else if( strncmp(buffer,(const char *) &"Imagen_KETA_MAX:", 16) ==0 )
            {
                Imagen.KETA_MAX=atof(&buffer[ 16]);
            }
            else if( strncmp(buffer,(const char *) &"Imagen_KETA_MIN:", 16) ==0 )
            {
                Imagen.KETA_MIN=atof(&buffer[ 16]);
            }
            else if( strncmp(buffer,(const char *) &"Escala_Invariantes_X:", 21) ==0 )
            {
                Imagen.Escala_Invariantes_X=atof(&buffer[ 21]);
            }
            else if( strncmp(buffer,(const char *) &"Escala_Invariantes_Y:", 21) ==0 )
            {
                Imagen.Escala_Invariantes_Y=atof(&buffer[ 21]);
            }

            else if( strncmp(buffer,(const char *) &"Imagen_Divisiones_X:", 20) ==0 )
            {
                for(n=0;n<Imagen.Divisiones_X;n++)
                {
                    free(Imagen.Cadena_X[ n]);
```



```
                }
                free(Imagen.Cadena_X);
                Imagen.Divisiones_X=atoi(&buffer[ 20]);
                Imagen.Puntos_Por_Division_X=
(Imagen.ancho-2*Imagen.MARGEN_X)/(Imagen.Divisiones_X-1);
                Imagen.Cadena_X=malloc(sizeof(char *)*Imagen.Divisiones_X);
                for(n=0;n<Imagen.Divisiones_X;n++)
                {
                    Imagen.Cadena_X[ n]=malloc(sizeof(char)*8);
                    sprintf((char *)Imagen.Cadena_X[ n],"10e%+d",n);
                }
            }
            else if( strncmp(buffer,(const char *) &"Imagen_Divisiones_Y:", 20) ==0 )
            {
                for(n=0;n<Imagen.Divisiones_Y;n++)
                {
                    free(Imagen.Cadena_Y[ n]);
                }
                free(Imagen.Cadena_Y);
                Imagen.Divisiones_Y=atoi(&buffer[ 20]);
                Imagen.Puntos_Por_Division_Y=(Imagen.alto-2*Imagen.MARGEN_Y)/(Imagen.Di
visiones_Y);
                Imagen.Cadena_Y=malloc(sizeof(char *)*Imagen.Divisiones_Y);
                for(n=0;n<Imagen.Divisiones_Y;n++)
                {
                    Imagen.Cadena_Y[ n]=malloc(sizeof(char)*8);
                    sprintf((char *)Imagen.Cadena_Y[ n],"10e%+d",1-n);
                }
            }


        }
        fclose(DATOS);

//      Esperamos a que todos los procesos terminen de leer
        MPI_Barrier(MPI_COMM_WORLD);
//      Borramos el fichero
        if(MPI.PROCESO_MPI==0)
        {
            if ((DATOS=fopen("Parametros.dat","w")) != NULL)
            {
                fclose(DATOS);
            }
        }
//      Si se modifican los parámetros de las imágenes recalcula lo necesario
        if(modifica_imagen)
        {
            Imagen.Puntos_Por_Division_X=
(Imagen.ancho-2*Imagen.MARGEN_X)/(Imagen.Divisiones_X-1);
            Imagen.Puntos_Por_Division_Y=(Imagen.alto-2*Imagen.MARGEN_Y)/(Imagen.Divisi
ones_Y);
            free(Imagen.buffer);
            Imagen.buffer= fftw_malloc(sizeof(unsigned char) * 9 * Imagen.ancho *
Imagen.alto);
            if(Imagen.buffer==NULL)
            {
                printf("\nError: No hay suficiente memoria disponible para la
generación de las imágenes\n");
                exit(1);
            }
            for (n=0;n< Imagen.alto;n++)
            {
                for (m=0;m<Imagen.ancho ;m++)
                {
                    Imagen.buffer[ (n*Imagen.ancho+m)*3]=255;
                    Imagen.buffer[ (n*Imagen.ancho+m)*3+1]=255;
                    Imagen.buffer[ (n*Imagen.ancho+m)*3+2]=255;
                }
```

```
            }
        }
    }
//  Escribimos los datos generales de salida para el programa ajustanavier
    if(MPI.PROCESO_MPI==0)
    {
        if ((DATOS=fopen("Parametros-salida.dat","w")) != NULL)
        {
            fprintf(DATOS,"VISCOSIDAD:%f\n",VISCOSIDAD);
            fprintf(DATOS,"dt:%f\n",dt);
            fprintf(DATOS,"K_FILTRO:%f\n",K_FILTRO);
            fprintf(DATOS,"CURSOR_K:%f\n",CURSOR_K);
            fprintf(DATOS,"CURSOR_E:%f\n",CURSOR_E);
            fprintf(DATOS,"TIEMPOTOTAL:%f\n",TIEMPOTOTAL);
            fprintf(DATOS,"KETA_ESTACIONARIA:%f\n",KETA_ESTACIONARIA);
            fprintf(DATOS,"FORZADO:%d\n",FORZADO);
            fprintf(DATOS,"CALCULAINVARIANTES:%d\n",CALCULAINVARIANTES);

            fclose(DATOS);
        }
    }
    return;
}



void WebEstado(void)
{
//  Genera una página web con los datos
    FILE *DATOS;
    if(GUARDAESTADO==0)return;
    if(MPI.PROCESO_MPI==0)
    {
        if ((DATOS=fopen("Graficas.html","w")) == NULL)
        {
            fprintf(stderr, "\nNo puedo abrir Graficas.html\n");
            exit(1);
        }

        fprintf(DATOS,"<html>\n<head><title>Gráficas</title>");
        fprintf(DATOS,
            "<meta name=\"GENERATOR\" content=\"Quanta Plus\">\n<meta
http-equiv=\"Content-Type\"content=\"text/html; charset=iso-8859-1\">\n<meta
http-equiv=\"Refresh\" content=\"2;Graficas.html\">\n"
        );
        fprintf(DATOS,"</head>\n<body>\n\n");
        if(CALCULAINVARIANTES)
        {
            fprintf(DATOS,"<table width=\"100%%\">\n<tbody>\n<tr>\n<td
width=\"50%%\"><h1>Espectro</h1></td>\n<td
width=\"50%%\"><h1>Jamón</h1></td>\n</tr>\n<tr>\n<td><a href=\"espectro.html\"><IMG
src=\"espectro.png\" name=\"imagen\"></td>\n<td><IMG src=\"invariantes.png\"
name=\"imagen\"></a></td>\n</tr>\n</tbody>\n</table>\n");
        }
        else
        {
            fprintf(DATOS,"<table width=\"100%%\">\n<tbody>\n<tr>\n<td
width=\"50%%\"><h1>Espectro</h1></td>\n<td
width=\"50%%\"><h1>Jamón</h1></td>\n</tr>\n<tr>\n<td><a href=\"espectro.html\"><IMG
src=\"espectro.png\" name=\"imagen\"></a></td>\n<td>No se calculan los
invariantes</td>\n</tr>\n</tbody>\n</table>\n");
        }


        fprintf(DATOS,"<table border=1>\n");
        fprintf(DATOS,"<tr><td>Navier-Stokes </td><td> (%dx%dx%d)  (%.1f,%.1f)</td><td
rowspan=\"4\"><a href=\"KETA.png\"><IMG height=150px
```



```
src=\"KETA.png\"name=\"imagen\"></a></td></tr>\n",N,N,N,K_FILTRO,N/2.0-1.5);
        fprintf(DATOS,"<tr><td>Tipo de integración: </td><td>
%s.</td></tr>\n",TIPOINT);
        fprintf(DATOS,"<tr><td>Tiempo de integración: %+.6e</td><td>Paso de
integración %e</td></tr>\n",TIEMPOTOTAL,dt);
        fprintf(DATOS,"<tr><td>Viscosidad= %+.6e</td><td>Viscosidad negativa=
%+.6e</td></tr>\n",VISCOSIDAD,VISCOSIDADNEG);

        fprintf(DATOS,"<tr><td>&#060;u_x&#062;= %+.6e</td><td>&#060;u_y&#062;=
%+.6e</td><td>&#060;u_z&#062;=
%+.6e</td></tr>\n",Estadisticas.u_Media_X,Estadisticas.u_Media_Y,Estadisticas.u_Media_Z
);
        fprintf(DATOS,"<tr><td>u'_x= %+.6e</td><td>u'_y= %+.6e</td><td>u'_z=
%+.6e</td></tr>\n",Estadisticas.u_Media_XX,Estadisticas.u_Media_YY,Estadisticas.u_Media
_ZZ);
        fprintf(DATOS,"<tr><td>&#060;u_x*u_y&#062;= %+.6e</td><td>&#060;u_y*u_z&#062;=
%+.6e</td><td>&#060;u_z*u_x&#062;=
%+.6e</td></tr>\n",Estadisticas.u_Media_XY,Estadisticas.u_Media_YZ,Estadisticas.u_Media
_ZX);

        #if (TIPO_KETA==KETA_VORTICIDAD)
    //      Cálculos con el EPSILON a partir de la vorticidad
            fprintf(DATOS,"<tr><td>EPSILON_vorticidad= %+.6e</td><td>ETA=
%+.6e</td><td>K*ETA_vorticidad= %+.6e</td></tr>\n",EPSILON, ETA, ETA*K_FILTRO);
        #elif (TIPO_KETA==KETA_ESPECTRO)
    //      Cálculos con el EPSILON a partir del espectro
            fprintf(DATOS,"<tr><td>EPSILON_espectro= %+.6e</td><td>ETA=
%+.6e</td><td>K*ETA_espectro= %+.6e</td></tr>\n",Espectro.EPSILON, ETA,
Estadisticas.KETA_espectro);

        #endif


        fprintf(DATOS,"\n<tr><td>Cuantos= %d / %d = %.2f%%</td><td>L=
%+.6e</td><td>Re_Lambda=
%+.6e</td></tr>",Estadisticas.cuantos,N3D_DIV,100.0*Estadisticas.cuantos/N3D_DIV,Espect
ro.L,Espectro.Re_lambda);
        fprintf(DATOS,"<table>\n");



        fprintf(DATOS,"\n\n</body>\n</html>\n");
        fclose(DATOS);
    }
    MPI_Barrier(MPI_COMM_WORLD);
}

fftw_real Modelo_E_k(fftw_real EPSILON_local,fftw_real K,fftw_real ETA_local,
fftw_real L_local)
{
//  Función que genera el modelo del espectro
    fftw_real E,aux;
    fftw_real C=1.5,BETA=5.2,C_ETA=0.4;

    fftw_real C_L=6.78,p_0=2.0;


    E=(C*pow(EPSILON_local,2.0/3.0)*pow(K,-5.0/3.0)*exp((-BETA)*(pow(pow(K*ETA_local,4)
+pow(C_ETA,4),1.0/4.0)-C_ETA)));

    //Multiplicamos por f_L
    aux=K*L_local;
    E*=pow((aux/pow(((aux*aux)+C_L),0.5)),5/3+p_0);

    return E;
}

fftw_real Numero_de_k(fftw_real K)
```

```
{
//  Devuelve el número de modos en las primeras rodajas del espectro
    fftw_real salida;
    if((K>0.5)&&(K<1.5))
        return 13.0;
    else if((K>=1.5)&&(K<2.5))
        return 37.0;
}

void RecomponeDatos(void)
{
//  Dado que el número de datos de los invariantes es enorme, se genera un archivo
temporar en el disco local de cada nodo y al final esta función lo recompone en uno
solo
    int n;
    char nombrefichero[ 30],buffer[ 4];
    FILE *DATOS,*Salida;
//  Hacemos un bucle que va proceso por proceso recomponiendo los ficheros
    if(GUARDAESTADO==0)return;
    for (n=0;n<MPI.NUMERO_DE_PROCESOS_MPI;n++)
    {
        if(n==MPI.PROCESO_MPI)
        {
            sprintf(nombrefichero,"temporal/invariantes-P%d.dat",MPI.PROCESO_MPI);
            if ((DATOS=fopen(nombrefichero,"rb")) == NULL)
            {
                fprintf(stderr, "\nNo puedo abrir %s\n",nombrefichero);
                exit(1);
            }

            if(MPI.PROCESO_MPI==0)
            {
                if ((Salida=fopen("invariantes.dat","wb")) == NULL)
                {
                    fprintf(stderr, "\nNo puedo abrir invariantes.dat\n");
                    exit(1);
                }
            }
            else
            {
                if ((Salida=fopen("invariantes.dat","ab")) == NULL)
                {
                    fprintf(stderr, "\nNo puedo abrir invariantes.dat\n");
                    exit(1);
                }
            }

            while(fread(&buffer,sizeof(fftw_real),1,DATOS))fwrite(&buffer,sizeof(fftw_r
eal),1,Salida);

            fclose(Salida);
            fclose(DATOS);
            unlink(nombrefichero);
        }

        MPI_Barrier(MPI_COMM_WORLD);

    }
    return;
}




void GuardaEstado(void)
{
//  Función que guarda el estado actual en disco
```



```
    char Nombre[ 30];

    MPI_File DATOS_MPI;
    int offset;

    if(GUARDAESTADO==0)return;  //Si no hay que guardar el estado se sale

    sprintf(Nombre,"MPI.Estado-N%d.dat",N);

    if(MPI.PROCESO_MPI==0)unlink(Nombre);
    MPI_Barrier(MPI_COMM_WORLD);
    if(MPI_File_open (MPI_COMM_WORLD, Nombre,  MPI_MODE_CREATE|MPI_MODE_RDWR,
MPI_INFO_NULL, &DATOS_MPI)!=MPI_SUCCESS)
    {
        if(MPI.PROCESO_MPI==0)
        {
            fprintf(stderr, "\nNo puedo abrir %s\n",Nombre);
        }
        exit(1);
    }


//  Escribimos primero la cabecera
    if(MPI.PROCESO_MPI==0)
    {
        MPI_File_write(DATOS_MPI, &N, sizeof(unsigned int) , MPI_CHARACTER,
&MPI.Estado);
        MPI_File_write(DATOS_MPI, &N3D, sizeof(unsigned int) , MPI_CHARACTER,
&MPI.Estado);
        MPI_File_write(DATOS_MPI, &N3D_DIV, sizeof(unsigned int) , MPI_CHARACTER,
&MPI.Estado);
        MPI_File_write(DATOS_MPI, &N3D_FREC, sizeof(unsigned int) , MPI_CHARACTER,
&MPI.Estado);

        MPI_File_write(DATOS_MPI, &K_FILTRO, sizeof(fftw_real) , MPI_CHARACTER,
&MPI.Estado);
        MPI_File_write(DATOS_MPI, &Kcuadrado_FILTRO, sizeof(fftw_real) ,
MPI_CHARACTER, &MPI.Estado);
        MPI_File_write(DATOS_MPI, &K_VISCOSIDAD, sizeof(fftw_real) , MPI_CHARACTER,
&MPI.Estado);
        MPI_File_write(DATOS_MPI, &K_VISCOSIDAD_CUAD, sizeof(fftw_real) ,
MPI_CHARACTER, &MPI.Estado);


        MPI_File_write(DATOS_MPI, &VISCOSIDAD, sizeof(fftw_real) , MPI_CHARACTER,
&MPI.Estado);
        MPI_File_write(DATOS_MPI, &VISCOSIDADNEG, sizeof(fftw_real) , MPI_CHARACTER,
&MPI.Estado);
        MPI_File_write(DATOS_MPI, &EPSILON, sizeof(fftw_real) , MPI_CHARACTER,
&MPI.Estado);
        MPI_File_write(DATOS_MPI, &ETA, sizeof(fftw_real) , MPI_CHARACTER,
&MPI.Estado);
        MPI_File_write(DATOS_MPI, &KETA_ESTACIONARIA, sizeof(fftw_real) ,
MPI_CHARACTER, &MPI.Estado);
        MPI_File_write(DATOS_MPI, &TIEMPOTOTAL, sizeof(fftw_real) , MPI_CHARACTER,
&MPI.Estado);
        MPI_File_write(DATOS_MPI, &dt, sizeof(fftw_real) , MPI_CHARACTER, &MPI.Estado);

        MPI_File_write(DATOS_MPI, &FORZADO, sizeof(unsigned int) , MPI_CHARACTER,
&MPI.Estado);
        MPI_File_write(DATOS_MPI, &CALCULAINVARIANTES, sizeof(unsigned int) ,
MPI_CHARACTER, &MPI.Estado);

        MPI_File_write(DATOS_MPI, &PID.error, sizeof(fftw_real) , MPI_CHARACTER,
&MPI.Estado);
        MPI_File_write(DATOS_MPI, &PID.deriv_error, sizeof(fftw_real) , MPI_CHARACTER,
&MPI.Estado);
        MPI_File_write(DATOS_MPI, &PID.int_error, sizeof(fftw_real) , MPI_CHARACTER,
```

```
&MPI.Estado);
        MPI_File_write(DATOS_MPI, &PID.error_ant, sizeof(fftw_real) , MPI_CHARACTER,
&MPI.Estado);
        MPI_File_write(DATOS_MPI, &PID.K_P, sizeof(fftw_real) , MPI_CHARACTER,
&MPI.Estado);
        MPI_File_write(DATOS_MPI, &PID.K_I, sizeof(fftw_real) , MPI_CHARACTER,
&MPI.Estado);
        MPI_File_write(DATOS_MPI, &PID.K_D, sizeof(fftw_real) , MPI_CHARACTER,
&MPI.Estado);

        MPI_File_write(DATOS_MPI, &Espectro.u_prima_cuadrado, sizeof(fftw_real) ,
MPI_CHARACTER, &MPI.Estado);
        MPI_File_write(DATOS_MPI, &Espectro.EPSILON, sizeof(fftw_real) ,
MPI_CHARACTER, &MPI.Estado);
        MPI_File_write(DATOS_MPI, &Espectro.L, sizeof(fftw_real) , MPI_CHARACTER,
&MPI.Estado);

        MPI_File_write(DATOS_MPI, &Imagen.alto, sizeof(int) , MPI_CHARACTER,
&MPI.Estado);
        MPI_File_write(DATOS_MPI, &Imagen.ancho, sizeof(int) , MPI_CHARACTER,
&MPI.Estado);
        MPI_File_write(DATOS_MPI, &Imagen.Divisiones_X, sizeof(int) , MPI_CHARACTER,
&MPI.Estado);
        MPI_File_write(DATOS_MPI, &Imagen.Divisiones_Y, sizeof(int) , MPI_CHARACTER,
&MPI.Estado);
        MPI_File_write(DATOS_MPI, &Imagen.MARGEN_X, sizeof(int) , MPI_CHARACTER,
&MPI.Estado);
        MPI_File_write(DATOS_MPI, &Imagen.MARGEN_Y, sizeof(int) , MPI_CHARACTER,
&MPI.Estado);
        MPI_File_write(DATOS_MPI, &Imagen.Puntos_Por_Division_X, sizeof(int) ,
MPI_CHARACTER, &MPI.Estado);
        MPI_File_write(DATOS_MPI, &Imagen.Puntos_Por_Division_Y, sizeof(int) ,
MPI_CHARACTER, &MPI.Estado);
        MPI_File_write(DATOS_MPI, &Imagen.KETA_MAX, sizeof(fftw_real) , MPI_CHARACTER,
&MPI.Estado);
        MPI_File_write(DATOS_MPI, &Imagen.KETA_MIN, sizeof(fftw_real) , MPI_CHARACTER,
&MPI.Estado);

    }

    MPI_File_close(&DATOS_MPI);

    MPI_Barrier(MPI_COMM_WORLD);

    if(MPI_File_open (MPI_COMM_WORLD, Nombre,  MPI_MODE_APPEND|MPI_MODE_RDWR,
MPI_INFO_NULL, &DATOS_MPI)!=MPI_SUCCESS)
    {
        if(MPI.PROCESO_MPI==0)
        {
            fprintf(stderr, "\nNo puedo abrir %s\n",Nombre);
        }
        exit(1);
    }

//  Escribimos el estado
    MPI_File_write_ordered(DATOS_MPI,VX.Frec,(TAM_LOCAL.NY*(N*(N/2+1)))*sizeof(fftw_com
plex), MPI_CHARACTER, &MPI.Estado);

    MPI_File_write_ordered(DATOS_MPI,VY.Frec,(TAM_LOCAL.NY*(N*(N/2+1)))*sizeof(fftw_com
plex), MPI_CHARACTER, &MPI.Estado);

    MPI_File_write_ordered(DATOS_MPI,VZ.Frec,(TAM_LOCAL.NY*(N*(N/2+1)))*sizeof(fftw_com
plex), MPI_CHARACTER, &MPI.Estado);

//  Escribimos VVORT
    MPI_File_write_ordered(DATOS_MPI,VVORTX.Frec,(TAM_LOCAL.NY*(N*(N/2+1)))*sizeof(fftw
_complex), MPI_CHARACTER, &MPI.Estado);
```



```
    MPI_File_write_ordered(DATOS_MPI,VVORTY.Frec,(TAM_LOCAL.NY*(N*(N/2+1)))*sizeof(fftw
_complex), MPI_CHARACTER, &MPI.Estado);

    MPI_File_write_ordered(DATOS_MPI,VVORTZ.Frec,(TAM_LOCAL.NY*(N*(N/2+1)))*sizeof(fftw
_complex), MPI_CHARACTER, &MPI.Estado);

//  Escribimos el estado nuevo
    MPI_File_write_ordered(DATOS_MPI,VX_nuevo.Frec,(TAM_LOCAL.NY*(N*(N/2+1)))*sizeof(ff
tw_complex), MPI_CHARACTER, &MPI.Estado);

    MPI_File_write_ordered(DATOS_MPI,VY_nuevo.Frec,(TAM_LOCAL.NY*(N*(N/2+1)))*sizeof(ff
tw_complex), MPI_CHARACTER, &MPI.Estado);

    MPI_File_write_ordered(DATOS_MPI,VZ_nuevo.Frec,(TAM_LOCAL.NY*(N*(N/2+1)))*sizeof(ff
tw_complex), MPI_CHARACTER, &MPI.Estado);

//  Por último el  VVORT anterior

    MPI_File_write_ordered(DATOS_MPI,VVORTX_ant.Frec,(TAM_LOCAL.NY*(N*(N/2+1)))*sizeof(
fftw_complex), MPI_CHARACTER, &MPI.Estado);

    MPI_File_write_ordered(DATOS_MPI,VVORTY_ant.Frec,(TAM_LOCAL.NY*(N*(N/2+1)))*sizeof(
fftw_complex), MPI_CHARACTER, &MPI.Estado);

    MPI_File_write_ordered(DATOS_MPI,VVORTZ_ant.Frec,(TAM_LOCAL.NY*(N*(N/2+1)))*sizeof(
fftw_complex), MPI_CHARACTER, &MPI.Estado);


    MPI_File_close(&DATOS_MPI);

    return;
}
```

```
/*************************************************************************
 *   Copyright (C) 2005 by Ignacio López Díaz                            *
 *   igna@us.es                                                          *
 *                                                                       *
 *   This program is free software; you can redistribute it and/or modify *
 *   it under the terms of the GNU General Public License as published by *
 *   the Free Software Foundation; either version 2 of the License, or   *
 *   (at your option) any later version.                                 *
 *                                                                       *
 *   This program is distributed in the hope that it will be useful,     *
 *   but WITHOUT ANY WARRANTY; without even the implied warranty of      *
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the       *
 *   GNU General Public License for more details.                        *
 *                                                                       *
 *   You should have received a copy of the GNU General Public License   *
 *   along with this program; if not, write to the                       *
 *   Free Software Foundation, Inc.,                                     *
 *   59 Temple Place - Suite 330, Boston, MA  02111-1307, USA.           *
 *************************************************************************/
#include "navier_mpi.h"



void Genera_Imagen_Espectro_Rodaja(void)
{
    int n;
    int Max;
#ifndef CONIMAGENES
    return;
#endif

    if(MPI.PROCESO_MPI==0)
    {
    //  Calculamos las potencias de los ejes
        Max=floor(log10(Estadisticas.MaxEspectro));

        for(n=0;n<Imagen.Divisiones_Y;n++)
        {
            sprintf((char *)Imagen.Cadena_Y[n],"10e%+d",Max-n);
        }
    //  Creamos los objetos para gestionar la imagen con imagemagick
        Imagen.W = NewMagickWand();
        Imagen.PW = NewPixelWand();
        Imagen.DW = NewDrawingWand();


        MagickConstituteImage ( Imagen.W, Imagen.ancho, Imagen.alto, "RGB", CharPixel,
Imagen.buffer);

    //  Dibujamos los ejes
        PixelSetColor( Imagen.PW, "#000000" );
        DrawSetStrokeWidth ( Imagen.DW, 2.0 );
        DrawSetStrokeColor( Imagen.DW, Imagen.PW );

        DrawLine( Imagen.DW, Imagen.MARGEN_X, Imagen.MARGEN_Y, Imagen.MARGEN_X,
Imagen.alto-Imagen.MARGEN_Y );
        DrawLine ( Imagen.DW, Imagen.MARGEN_X, Imagen.alto-Imagen.MARGEN_Y,
Imagen.ancho-Imagen.MARGEN_X, Imagen.alto-Imagen.MARGEN_Y);

        for(n=0;n<Imagen.Divisiones_X;n++)
        {
            PixelSetColor( Imagen.PW, "#000000" );
            DrawSetStrokeWidth ( Imagen.DW, 1.0 );
            DrawSetStrokeColor( Imagen.DW, Imagen.PW );
            DrawAnnotation(Imagen.DW,Imagen.MARGEN_X+n*Imagen.Puntos_Por_Division_X,
Imagen.alto-Imagen.MARGEN_Y/2,Imagen.Cadena_X[n]);
```



```
            PixelSetColor( Imagen.PW, "#9090f0" );
            DrawSetStrokeWidth ( Imagen.DW, 1.0 );
            DrawSetStrokeColor( Imagen.DW, Imagen.PW );
            if(n>0)DrawLine ( Imagen.DW,
Imagen.MARGEN_X+n*Imagen.Puntos_Por_Division_X, Imagen.MARGEN_Y ,
Imagen.MARGEN_X+n*Imagen.Puntos_Por_Division_X, Imagen.alto-Imagen.MARGEN_Y);
        }

        for(n=0;n<Imagen.Divisiones_Y;n++)
        {
            PixelSetColor( Imagen.PW, "#000000" );
            DrawSetStrokeWidth ( Imagen.DW, 1.0 );
            DrawSetStrokeColor( Imagen.DW, Imagen.PW );
            DrawAnnotation(Imagen.DW,10,Imagen.MARGEN_Y+
(Imagen.alto-2*Imagen.MARGEN_Y)/(Imagen.Divisiones_Y)+n*Imagen.Puntos_Por_Division_Y,Im
agen.Cadena_Y[n]);
            PixelSetColor( Imagen.PW, "#9090f0" );
            DrawSetStrokeWidth ( Imagen.DW, 1.0 );
            DrawSetStrokeColor( Imagen.DW, Imagen.PW );
            DrawLine ( Imagen.DW,
                Imagen.MARGEN_X,
                Imagen.MARGEN_Y+
(Imagen.alto-2*Imagen.MARGEN_Y)/(Imagen.Divisiones_Y)+n*Imagen.Puntos_Por_Division_Y,
                Imagen.ancho-Imagen.MARGEN_X,
                Imagen.MARGEN_Y+(Imagen.alto-2*Imagen.MARGEN_Y)/(Imagen.Divisiones_Y)+n
*Imagen.Puntos_Por_Division_Y);
        }

//      Dibujamos las líneas de los espectros
        for(n=1;n<N/2;n++)
        {

            if( (finite( log10(Espectro.Valores_Rodaja[n])  )) && (finite(
log10(Espectro.Valores_Rodaja[n+1])  )) )
            {
                //E(k)
                PixelSetColor( Imagen.PW, "#0000ff" );
                DrawSetStrokeWidth ( Imagen.DW, 2.0 );
                DrawSetStrokeColor( Imagen.DW, Imagen.PW );

                DrawLine ( Imagen.DW,
                (int)(Imagen.Puntos_Por_Division_X*log10(n)+Imagen.MARGEN_X),
                (int)(Imagen.MARGEN_Y+(Imagen.alto-2*Imagen.MARGEN_Y)/(Imagen.Divisione
s_Y)-rint(Imagen.Puntos_Por_Division_Y*(log10(Espectro.Valores_Rodaja[n])-Max) )),
                (int)(Imagen.Puntos_Por_Division_X*log10(n+1)+Imagen.MARGEN_X),
                (int)(Imagen.MARGEN_Y+(Imagen.alto-2*Imagen.MARGEN_Y)/(Imagen.Divisione
s_Y)-rint(Imagen.Puntos_Por_Division_Y*(log10(Espectro.Valores_Rodaja[n+1])-Max) ))
                );

    //          //E(k)* k^(5/3)/ EPSILON^(2/3)

                PixelSetColor( Imagen.PW, "#00ff00" );
                DrawSetStrokeWidth ( Imagen.DW, 2.0 );
                DrawSetStrokeColor( Imagen.DW, Imagen.PW );

                DrawLine ( Imagen.DW,
                (int)(Imagen.Puntos_Por_Division_X*log10(n)+Imagen.MARGEN_X),
                (int)(Imagen.MARGEN_Y+(Imagen.alto-2*Imagen.MARGEN_Y)/(Imagen.Divisione
s_Y)-rint(Imagen.Puntos_Por_Division_Y*(log10(pow(pow(n,5)/(EPSILON*EPSILON),1.0/3.0)*E
spectro.Valores_Rodaja[n])-Max) )),
                (int)(Imagen.Puntos_Por_Division_X*log10(n+1)+Imagen.MARGEN_X),
                (int)(Imagen.MARGEN_Y+(Imagen.alto-2*Imagen.MARGEN_Y)/(Imagen.Divisione
s_Y)-rint(Imagen.Puntos_Por_Division_Y*(log10(pow(pow(n+1,5)/(EPSILON*EPSILON),1.0/3.0)
*Espectro.Valores_Rodaja[n+1])-Max) ))
                );
```

```
    //          Modelos

                PixelSetColor( Imagen.PW, "#ffff00" );
                DrawSetStrokeWidth ( Imagen.DW, 2.0 );
                DrawSetStrokeColor( Imagen.DW, Imagen.PW );

                DrawLine ( Imagen.DW,
                        (int)(Imagen.Puntos_Por_Division_X*log10(n)+Imagen.MARGEN_X),
                        (int)(Imagen.MARGEN_Y+(Imagen.alto-2*Imagen.MARGEN_Y)/(Imagen.
Divisiones_Y)-rint(Imagen.Puntos_Por_Division_Y*(log10(Modelo_E_k(EPSILON_ESTACIONARIO,
n,ETA_ESTACIONARIA,Espectro.L))-Max) )),
                        (int)(Imagen.Puntos_Por_Division_X*log10(n+1)+Imagen.MARGEN_X)
,
                        (int)(Imagen.MARGEN_Y+(Imagen.alto-2*Imagen.MARGEN_Y)/(Imagen.
Divisiones_Y)-rint(Imagen.Puntos_Por_Division_Y*(log10(Modelo_E_k(EPSILON_ESTACIONARIO,
n+1,ETA_ESTACIONARIA,Espectro.L))-Max) ))
                        );

                PixelSetColor( Imagen.PW, "#999900" );
                DrawSetStrokeWidth ( Imagen.DW, 2.0 );
                DrawSetStrokeColor( Imagen.DW, Imagen.PW );
                DrawLine ( Imagen.DW,
                        (int)(Imagen.Puntos_Por_Division_X*log10(n)+Imagen.MARGEN_X),
                        (int)(Imagen.MARGEN_Y+(Imagen.alto-2*Imagen.MARGEN_Y)/(Imagen.
Divisiones_Y)-rint(Imagen.Puntos_Por_Division_Y*(log10(Modelo_E_k(EPSILON,n,ETA,Espectr
o.L))-Max) )),
                        (int)(Imagen.Puntos_Por_Division_X*log10(n+1)+Imagen.MARGEN_X)
,
                        (int)(Imagen.MARGEN_Y+(Imagen.alto-2*Imagen.MARGEN_Y)/(Imagen.
Divisiones_Y)-rint(Imagen.Puntos_Por_Division_Y*(log10(Modelo_E_k(EPSILON,n+1,ETA,Espec
tro.L))-Max) ))
                        );


            }
            else
            {
                if(finite( log10(Espectro.Valores_Rodaja[ n] )  ))
                {
                    DrawPoint ( Imagen.DW,
                                (int)(Imagen.Puntos_Por_Division_X*log10(n)+Imagen.MARG
EN_X),
                                (int)(Imagen.MARGEN_Y+(Imagen.alto-2*Imagen.MARGEN_Y)/(
Imagen.Divisiones_Y)-rint(Imagen.Puntos_Por_Division_Y*(log10(Espectro.Valores_Rodaja[ n
])-Max) )));
                }
            }
        }

    //  Dibujamos las líneas de cursor
        PixelSetColor( Imagen.PW, "#f090f0" );
        DrawSetStrokeWidth ( Imagen.DW, 1.0 );
        DrawSetStrokeColor( Imagen.DW, Imagen.PW );
        DrawLine ( Imagen.DW,
        (int)(Imagen.Puntos_Por_Division_X*log10(CURSOR_K)+Imagen.MARGEN_X),
        Imagen.MARGEN_Y ,
        (int)(Imagen.Puntos_Por_Division_X*log10(CURSOR_K)+Imagen.MARGEN_X),
        Imagen.alto-Imagen.MARGEN_Y);

        DrawLine ( Imagen.DW,
        (int)(Imagen.MARGEN_X),
        (int)(Imagen.MARGEN_Y+(Imagen.alto-2*Imagen.MARGEN_Y)/(Imagen.Divisiones_Y)-rin
t(Imagen.Puntos_Por_Division_Y*(log10(CURSOR_E)-Max) )),
        (int)(Imagen.ancho-Imagen.MARGEN_X),
        (int)(Imagen.MARGEN_Y+(Imagen.alto-2*Imagen.MARGEN_Y)/(Imagen.Divisiones_Y)-rin
t(Imagen.Puntos_Por_Division_Y*(log10(CURSOR_E)-Max) ))
        );
```



```
    //  Dibujamos la línea que limita la inyección de energía
        PixelSetColor( Imagen.PW, "#ff0000" );
        DrawSetStrokeWidth ( Imagen.DW, 1.0 );
        DrawSetStrokeColor( Imagen.DW, Imagen.PW );
        DrawLine ( Imagen.DW,
                (int)(Imagen.Puntos_Por_Division_X*log10(K_VISCOSIDAD)+Imagen.MARGEN_X)
,
                Imagen.MARGEN_Y ,
                (int)(Imagen.Puntos_Por_Division_X*log10(K_VISCOSIDAD)+Imagen.MARGEN_X)
,
                Imagen.alto-Imagen.MARGEN_Y);

//      Guardamos la imagen en disco y liberamos la memoria
        MagickDrawImage( Imagen.W, Imagen.DW );

        MagickSetImageType( Imagen.W, TrueColorType );
        MagickSetImageDepth( Imagen.W, 8 );

        MagickWriteImages(Imagen.W,"espectro.png",MagickTrue);
        DestroyPixelWand( Imagen.PW );
        DestroyDrawingWand( Imagen.DW );
        DestroyMagickWand( Imagen.W );
    }

    return ;
}



void Inicializa_Imagen_Invariantes(void)
{
//  Función que realiza las operaciones necesarias para iniciar una imagen de
invariantes nueva
#ifndef CONIMAGENES
    return;
#endif
    Estadisticas.cuantos=0;
    return;
}


void Genera_Imagen_Invariantes(void)
{
//  Función que genera una imagen de invariantes después de haber calculado los puntos
    int n,m;
    FILE *DATOS;
#ifndef CONIMAGENES
    return;
#endif

//  Creamos los objetos para gestionar la imagen con imagemagick
    Imagen.W = NewMagickWand();
    Imagen.PW = NewPixelWand();
    Imagen.DW = NewDrawingWand();

//  Recuperamos en el proceso cero todos los puntos de la imagen
    MPI_Barrier(MPI_COMM_WORLD);

    MPI_Allreduce ( Imagen.buffer_MPI,Imagen.buffer_MPI2, Imagen.ancho*Imagen.alto,
MPI_INT,MPI_SUM, MPI_COMM_WORLD);
//  Los pasamos al mapa de colores
    if(MPI.PROCESO_MPI==0)
    {
        for (n=0;n< Imagen.alto;n++)
        {
            for (m=0;m<Imagen.ancho ;m++)
            {
```

```
                if( Imagen.buffer_MPI2[ n*Imagen.ancho+m]>30
)Imagen.buffer_MPI2[ n*Imagen.ancho+m]=30;
                if( Imagen.buffer_MPI2[ n*Imagen.ancho+m]>0 )
                {
                    if( Imagen.buffer_MPI2[ n*Imagen.ancho+m]<10 )
                    {
                        Imagen.buffer[ (n*Imagen.ancho+m)*3]=(unsigned
char)(255-Imagen.buffer_MPI2[ n*Imagen.ancho+m]*25);
                        Imagen.buffer[ (n*Imagen.ancho+m)*3+1]=(unsigned
char)(255-Imagen.buffer_MPI2[ n*Imagen.ancho+m]*25);
                        Imagen.buffer[ (n*Imagen.ancho+m)*3+2]=(unsigned
char)(255-Imagen.buffer_MPI2[ n*Imagen.ancho+m]*25);
                    }
                    else if( Imagen.buffer_MPI2[ n*Imagen.ancho+m]<20 )
                    {
                        Imagen.buffer[ (n*Imagen.ancho+m)*3]=(unsigned char)0;
                        Imagen.buffer[ (n*Imagen.ancho+m)*3+1]=(unsigned char)0;
                        Imagen.buffer[ (n*Imagen.ancho+m)*3+2]=(unsigned
char)((Imagen.buffer_MPI2[ n*Imagen.ancho+m]-10)*25);
                    }
                    else
                    {
                        Imagen.buffer[ (n*Imagen.ancho+m)*3]=(unsigned char)0;
                        Imagen.buffer[ (n*Imagen.ancho+m)*3+1]=(unsigned
char)((Imagen.buffer_MPI2[ n*Imagen.ancho+m]-20)*25);
                        Imagen.buffer[ (n*Imagen.ancho+m)*3+2]=(unsigned char)255;
                    }
                }
                else
                {
                    Imagen.buffer[ (n*Imagen.ancho+m)*3]=(unsigned char)255;
                    Imagen.buffer[ (n*Imagen.ancho+m)*3+1]=(unsigned char)255;
                    Imagen.buffer[ (n*Imagen.ancho+m)*3+2]=(unsigned char)255;
                }
            }
        }

        MagickConstituteImage ( Imagen.W, Imagen.ancho, Imagen.alto, "RGB", CharPixel,
Imagen.buffer);

    //  Dibujamos los ejes
        PixelSetColor( Imagen.PW, "#00ff00" );
        DrawSetStrokeWidth ( Imagen.DW, 1.0 );
        DrawSetStrokeColor( Imagen.DW, Imagen.PW );

        DrawLine( Imagen.DW, Imagen.ancho/2, 0, Imagen.ancho/2, Imagen.alto-1);
        DrawLine ( Imagen.DW, 0, Imagen.alto/2, Imagen.ancho-1, Imagen.alto/2);

//      Guardamos la imagen en disco
        MagickDrawImage( Imagen.W, Imagen.DW );

        MagickSetImageType( Imagen.W, TrueColorType );
        MagickSetImageDepth( Imagen.W, 8 );

        MagickWriteImages(Imagen.W,"invariantes.png",MagickTrue);

    }

//  Borramos la imagen para los próximos cálculos
    for (n=0;n< Imagen.alto;n++)
    {
        for (m=0;m<Imagen.ancho ;m++)
        {
            Imagen.buffer[ (n*Imagen.ancho+m)*3]=255;
            Imagen.buffer[ (n*Imagen.ancho+m)*3+1]=255;
            Imagen.buffer[ (n*Imagen.ancho+m)*3+2]=255;
            Imagen.buffer_MPI[ n*Imagen.ancho+m]=0;
        }
```



```
    }   DestroyPixelWand( Imagen.PW );
//  Liberamos la memoria
    DestroyDrawingWand( Imagen.DW );
    DestroyMagickWand( Imagen.W );
    MPI_Barrier(MPI_COMM_WORLD);
    return;
}



void Poner_Punto_Imagen_Invariantes(fftw_real determinante,fftw_real adjuntos)
{
//  Esta función pone un punto en la imagen de los invariantes
    int n,m;
#ifndef CONIMAGENES
    return;
#endif
    m=Imagen.ancho/2+(Imagen.ancho/2)*(Imagen.Escala_Invariantes_X*determinante);
    n=Imagen.alto/2-(Imagen.alto/2)*(Imagen.Escala_Invariantes_Y*adjuntos);

    if( (n<Imagen.alto)&&(n>=0) && (m<Imagen.ancho)&&(m>=0) )
    {
        if(Imagen.buffer_MPI[ n*Imagen.ancho+m]<30)
            Imagen.buffer_MPI[ n*Imagen.ancho+m]++;
    }
    else
        Estadisticas.cuantos++;

    return;
}


void Genera_Imagen_KETA(fftw_real KETA,fftw_real KETA_espectro)
{
//  Genera una imagen de la evolución de KETA en el último tiempo integral
    int n;
    unsigned char KETACAD[30];
#ifndef CONIMAGENES
    return;
#endif
    if(MPI.PROCESO_MPI==0)
    {
        PID.KETA[PID.indice_KETA]=KETA;
        PID.KETA_espectro[PID.indice_KETA]=KETA_espectro;

        while(PID.Periodo_KETA<=TIEMPOTOTAL)
        {
            PID.Periodo_KETA+=1.0/PID.tam_KETA;
            PID.indice_KETA=(PID.indice_KETA+1)%PID.tam_KETA;
            PID.KETA[PID.indice_KETA]=KETA;
            PID.KETA_espectro[PID.indice_KETA]=KETA_espectro;
        }
    //  Creamos los objetos para gestionar la imagen con imagemagick
        Imagen.W = NewMagickWand();
        Imagen.PW = NewPixelWand();
        Imagen.DW = NewDrawingWand();

        MagickConstituteImage ( Imagen.W, Imagen.ancho, Imagen.alto, "RGB", CharPixel,
Imagen.buffer);

    //  Dibujamos los ejes
        PixelSetColor( Imagen.PW, "#000000" );
        DrawSetStrokeWidth ( Imagen.DW, 3.0 );
        DrawSetStrokeColor( Imagen.DW, Imagen.PW );

        DrawLine( Imagen.DW, Imagen.MARGEN_X, Imagen.MARGEN_Y, Imagen.MARGEN_X,
Imagen.alto-Imagen.MARGEN_Y );
        DrawLine ( Imagen.DW, Imagen.MARGEN_X, Imagen.alto-Imagen.MARGEN_Y,
```

```
Imagen.ancho-Imagen.MARGEN_X, Imagen.alto-Imagen.MARGEN_Y);
        DrawSetStrokeWidth ( Imagen.DW, 1.0 );
        sprintf((char *)KETACAD,"%.3f",Imagen.KETA_MIN);
        DrawAnnotation(Imagen.DW,10,Imagen.alto-Imagen.MARGEN_Y,KETACAD);
        #if (TIPO_KETA==KETA_VORTICIDAD)
    //      Calculo con el EPSILON a partir de la vorticidad
            sprintf((char *)KETACAD,"PID: EPSILON(Vorticidad)");
        #elif (TIPO_KETA==KETA_ESPECTRO)
    //      Calculo con el EPSILON a partir del espectro
            sprintf((char *)KETACAD,"PID: EPSILON(Espectro)");
        #endif
        DrawSetFontSize ( Imagen.DW, 28.0);
        DrawAnnotation(Imagen.DW,2*Imagen.MARGEN_X,Imagen.MARGEN_Y,KETACAD);
        DrawSetFontSize ( Imagen.DW, 16.0);

            PixelSetColor( Imagen.PW, "#0000FF" );
            DrawSetStrokeWidth ( Imagen.DW, 1.0 );
            DrawSetStrokeColor( Imagen.DW, Imagen.PW );
            sprintf((char *)KETACAD,"%.3f",KETA_ESTACIONARIA);
            DrawAnnotation(Imagen.DW,10,
            (Imagen.alto-Imagen.MARGEN_Y)-
            (Imagen.alto-Imagen.MARGEN_Y)*((KETA_ESTACIONARIA-Imagen.KETA_MIN)/((Imagen
.KETA_MAX+Imagen.KETA_MIN)/1.0)),
            KETACAD);
            PixelSetColor( Imagen.PW, "#9090f0" );
            DrawSetStrokeWidth ( Imagen.DW, 1.0 );
            DrawSetStrokeColor( Imagen.DW, Imagen.PW );

            DrawLine ( Imagen.DW,
                Imagen.MARGEN_X,
                (Imagen.alto-Imagen.MARGEN_Y)-(Imagen.alto-Imagen.MARGEN_Y)*((KETA_ESTA
CIONARIA-Imagen.KETA_MIN)/(Imagen.KETA_MAX-Imagen.KETA_MIN)),
                Imagen.ancho-Imagen.MARGEN_X,
                (Imagen.alto-Imagen.MARGEN_Y)-(Imagen.alto-Imagen.MARGEN_Y)*((KETA_ESTA
CIONARIA-Imagen.KETA_MIN)/(Imagen.KETA_MAX-Imagen.KETA_MIN)));

    //  Pintamos KETA
        PixelSetColor( Imagen.PW, "#00FF00" );
        DrawSetStrokeWidth ( Imagen.DW, 2.0 );
        DrawSetStrokeColor( Imagen.DW, Imagen.PW );

        for(n=0;n<PID.tam_KETA-1;n++)
        {
            DrawLine ( Imagen.DW,
                (Imagen.ancho-2*Imagen.MARGEN_X)*n/PID.tam_KETA+Imagen.MARGEN_X,
               (Imagen.alto-Imagen.MARGEN_Y)-((Imagen.alto-Imagen.MARGEN_Y)*((PID.KETA[ ((n
+PID.indice_KETA)%PID.tam_KETA)]-Imagen.KETA_MIN)/(Imagen.KETA_MAX-Imagen.KETA_MIN))),
                (Imagen.ancho-2*Imagen.MARGEN_X)*(n+1)/PID.tam_KETA+Imagen.MARGEN_X,
                (Imagen.alto-Imagen.MARGEN_Y)-((Imagen.alto-Imagen.MARGEN_Y)*((PID.KETA
[ ((n+1+PID.indice_KETA)%PID.tam_KETA)]-Imagen.KETA_MIN)/(Imagen.KETA_MAX-Imagen.KETA_MI
N)))
                );
        }
    //  Pintamos KETA_espectro
        PixelSetColor( Imagen.PW, "#0000FF" );
        DrawSetStrokeWidth ( Imagen.DW, 2.0 );
        DrawSetStrokeColor( Imagen.DW, Imagen.PW );

        for(n=0;n<PID.tam_KETA-1;n++)
        {
            DrawLine ( Imagen.DW,
                (Imagen.ancho-2*Imagen.MARGEN_X)*n/PID.tam_KETA+Imagen.MARGEN_X,
               (Imagen.alto-Imagen.MARGEN_Y)-((Imagen.alto-Imagen.MARGEN_Y)*((PID.KETA_esp
ectro[ ((n+PID.indice_KETA)%PID.tam_KETA)]-Imagen.KETA_MIN)/(Imagen.KETA_MAX-Imagen.KETA
_MIN))),
                (Imagen.ancho-2*Imagen.MARGEN_X)*(n+1)/PID.tam_KETA+Imagen.MARGEN_X,
                (Imagen.alto-Imagen.MARGEN_Y)-((Imagen.alto-Imagen.MARGEN_Y)*((PID.KETA
_espectro[ ((n+1+PID.indice_KETA)%PID.tam_KETA)]-Imagen.KETA_MIN)/(Imagen.KETA_MAX-Image
```



```
n.KETA_MIN)))
                );
        }

//      Guardamos la imagen en disco y liberamos la memoria
        MagickDrawImage( Imagen.W, Imagen.DW );

        MagickSetImageType( Imagen.W, TrueColorType );
        MagickSetImageDepth( Imagen.W, 8 );

        MagickWriteImages(Imagen.W,"KETA.png",MagickTrue);

        DestroyPixelWand( Imagen.PW );
        DestroyDrawingWand( Imagen.DW );
        DestroyMagickWand( Imagen.W );
    }
    return ;
}
```