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 ;
}
```