Apéndices del Trabajo de Fin de Máster

Guillermo Díez Señorans

28 de noviembre de 2017

Traslocación de polímeros a través de nanoporos

1. Estimación del radio de giro del polímero

El *radio de giro* (R_g) es una medida del tamaño de un polímero en equilibrio; se define como:

$$R_g^2 = \frac{1}{N}\sum (\mathbf{r}_i - \mathbf{R}_{cm})^2$$

El cálculo analítico de esta magnitud se complica rápidamente con la complejidad del polímero, por lo que en primer lugar aproximaremos un modelo polimérico *WLCM* (*Worm Like Chain Model*, polímero con interacción a tres partículas mediante un potencial de flexión) a un polímero ideal mediante la *longitud de persistencia* (l_p), definida como la distancia a la que decae típicamente la correlación del vector tangente entre dos puntos de una hebra:

$$l_p :< \hat{e}(x)\cdot\hat{e}(y) >= e^{-\frac{|x-y|}{l_p}}$$

Dado que en distancias $\sim l_p$ el polímero no se flexiona, se puede identificar un polímero *WLCM* de longitud L y $N = L/b_0$ centros de interacción con un polímero ideal de $N' = L/l_p$ partículas (fig. 1).

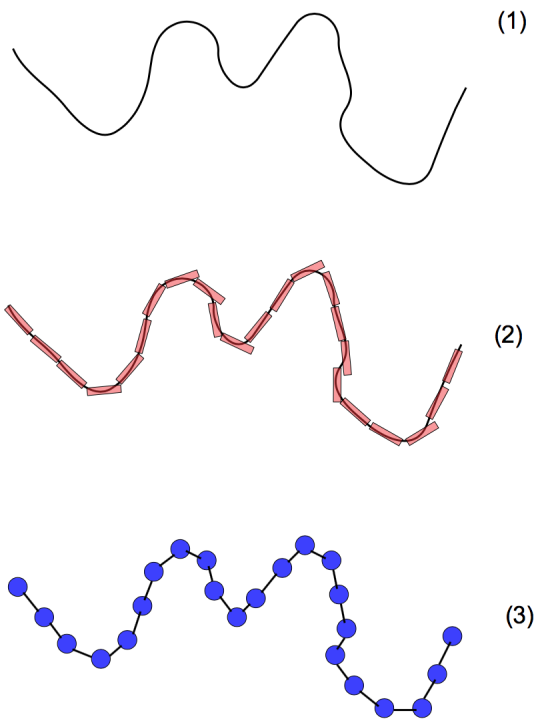

Figura 1: Esquema de la simplificación $WLCM \rightarrow$ polímero ideal mediante el uso de la longitud de persistencia l_p, esquematizada en los rectángulos de (2)

Adicionalmente utilizaremos una aproximación de pequeño ángulo ($i.e\ k_\theta \gg k_B T$), de forma que la energía del polímero será:

$$V_T = \frac{k_\theta}{2} \sum \theta_i^2 \qquad \text{Energía potencial}$$

Donde θ_i es el ángulo que ocupa la partícula i-ésima respecto del eje $(\overrightarrow{r}_{i-1} - \overrightarrow{r}_{i-2})$; cada microestado de un polímero de N partículas se puede caracterizar especificando el ángulo sólido (Ω) que ocupa cada partícula respecto del eje $(\overrightarrow{r}_{i-1} - \overrightarrow{r}_{i-2})$, esto es mediante una lista de N valores: $\{\Omega_i\}_{i=1}^{N}$

Llamando $\beta \equiv {}^1\!/_{k_B T}$, $E_i \equiv$ energía del *i-ésimo* microestado, $E_i = \frac{k_\theta}{2} \sum\limits_i \theta_i^2$,

La función de partición del conjunto canónico (Z) será:

$$Z = \sum_{\{\Omega_i\}} e^{-\beta E_i} = \sum_{\Omega_1, \Omega_2, \ldots \Omega_N} e^{-\beta \frac{k_\theta}{2} \sum\limits_i \theta_i^2} = \sum_{\Omega_1} \sum_{\Omega_2} \cdots \sum_{\Omega_N} \prod_{i=1}^{N} e^{-\beta \frac{k_\theta}{2} \theta_i^2} =$$

$$= \sum_{\theta_1} e^{-\beta \frac{k_\theta}{2} \theta_1^2} \sum_{\Omega_2} e^{-\beta \frac{k_\theta}{2} \theta_2^2} \cdots \sum_{\Omega_N} e^{-\beta \frac{k_\theta}{2} \theta_N^2} \equiv z^N$$

Donde $z = \sum\limits_{\Omega} e^{-\beta \frac{k_\theta}{2} \theta^2}$, $\theta \in [0, \pi]$. Convirtiendo el sumatorio z en la integral:

$$z \rightarrow \iint_{\mathbf{S^1}} e^{-\beta \frac{k_\theta}{2} \theta^2} D(\Omega)\, d\Omega \tag{1}$$

Siendo $D(\Omega) \equiv \frac{\delta n}{\delta \Omega}$ la densidad de estados (en este caso $D(\Omega) = 1$, dado que cada estado se caracteriza precisamente con un punto de la esfera unidad, *i.e* un valor de la cantidad Ω):

$$z = \int_{-\pi}^{\pi} d\phi \int_{0}^{\pi} \sin(\theta) e^{-\beta \frac{k_\theta}{2} \theta^2} \, d\theta = 2\pi \int_{0}^{\pi} \sin(\theta) e^{-\beta \frac{k_\theta}{2} \theta^2} \, d\theta$$

Para calcular la integral (1) introduzco la cantidad
Aplicando esta definición a dos partículas contiguas del polímero:

$$\hat{e}(x_i) \equiv \hat{e}_i = \frac{\overrightarrow{r}_{i+1} - \overrightarrow{r}_i}{|\overrightarrow{r}_{i+1} - \overrightarrow{r}_i|} = \frac{\overrightarrow{r}_{i+1} - \overrightarrow{r}_i}{b_0}$$

$$< \hat{e}_{i-1} \cdot \hat{e}_i >= e^{-\frac{b_0}{l_p}} =< \cos(\theta) >$$

La variable aleatoria θ se distribuirá de acuerdo con la distribución de Maxwell-Boltzmann:

$$P_\Omega = \frac{e^{-\beta \frac{k_\theta}{2} \theta_\Omega^2}}{z} = \frac{e^{-\beta \frac{k_\theta}{2} \theta^2}}{2\pi \int_0^\pi \sin(\theta) e^{-\beta \frac{k_\theta}{2} \theta^2} \, d\theta}$$

de modo que:

$$< \cos(\theta) >= \iint_{\mathbf{S^1}} P_\Omega \cos(\theta) \, d\Omega = \frac{2\pi \int_0^\pi e^{-\beta \frac{k_\theta}{2} \theta^2} \cos(\theta) \, \sin(\theta) \, d\theta}{2\pi \int_0^\pi \sin(\theta) e^{-\beta \frac{k_\theta}{2} \theta^2} \, d\theta}$$

Utilizando ahora una aproximación de pequeños ángulos:

$$\left.\begin{array}{ll} \text{(i)} & {}^{k_\theta}/_{k_B} = \beta k_\theta \gg 1 \quad \text{(i)} \\ \text{(ii)} & \sin(\theta) \approx \theta \quad \text{(ii)} \\ \text{(iii)} & \cos(\theta) \approx 1 - \frac{\theta^2}{2} \quad \text{(iii)} \end{array}\right\} \longrightarrow < \cos(\theta) > \simeq \frac{\int_0^\infty e^{-\beta \frac{k_\theta}{2} \theta^2} \left(\theta - \frac{1}{2}\theta^3\right) \, d\theta}{\int_0^\infty \theta e^{-\beta \frac{k_\theta}{2} \theta^2} \, d\theta} =$$

$$= 1 - \frac{1}{2} \beta k_\theta \int_0^\infty e^{-\beta \frac{k_\theta}{2} \theta^2} \theta^3 \, d\theta = 1 - \frac{1}{\beta k_\theta}$$

Y de acuerdo con la definición de longitud de persistencia:

$$< \cos(\theta) >= \frac{\beta k_\theta - 1}{\beta k_\theta} = e^{-\frac{b_0}{l_p}} \longrightarrow \log\left(\frac{\beta k_\theta - 1}{\beta k_\theta}\right) = -\frac{b_0}{l_p}$$

$$\left[l_p = \frac{b_0}{\log\left(\frac{\beta k_\theta}{\beta k_\theta - 1}\right)} \simeq b_0 \frac{k_B T}{k_\theta} \right]$$

A continuación se deriva la expresión para $\sqrt{\langle R_g^2 \rangle}$ del polímero ideal (de $N = L/l_p$ partículas y longitud entre monómeros l_p); para ello se utilizará la misma noción de microestado que en el cálculo anterior.

El radio de giro se puede expresar:

$$R_g^2 = \frac{1}{N} \sum (\mathbf{r}_i - \mathbf{R}_{cm})^2 = \frac{1}{N} \sum \left(\mathbf{r}_i^2 + \mathbf{R}_{cm}^2 - 2\mathbf{r}_i \mathbf{R}_{cm}\right) =$$

$$= \frac{1}{N}\sum \mathbf{r}_i^2 + \mathbf{R}_{cm}^2 - \frac{2}{N}\mathbf{R}_{cm}\sum \mathbf{r}_i = \frac{1}{N}\sum \mathbf{r}_i^2 - \mathbf{R}_{cm}^2$$

Escribiendo el radiovector de la i-ésima partícula como la suma del camino desde la primera, $\mathbf{r}_i = \sum_{j=1}^{i} \Delta\mathbf{r}_j$, donde $\Delta\mathbf{r}_j = \mathbf{r}_j - \mathbf{r}_{j-1}$ se tiene:

$$\mathbf{r}_i^2 = \sum_{j,j'=1}^{i,i} \Delta\mathbf{r}_j \Delta\mathbf{r}_{j'}$$

$$\mathbf{R}_{cm} = \frac{1}{N}\sum_i \mathbf{r}_i = \frac{1}{N}\sum_i \sum_{j=1}^{i} \Delta\mathbf{r}_j \rightarrow \mathbf{R}_{cm}^2 = \frac{1}{N^2}\sum_i \sum_{j=1}^{i} \Delta\mathbf{r}_j \sum_{i'} \sum_{j'=1}^{i'} \Delta\mathbf{r}_{j'}$$

$$\mathbf{R}_{cm}^2 = \frac{1}{N^2}\sum_{i,i'=1}^{N,N} \sum_{j,j'=1}^{i,i'} \Delta\mathbf{r}_j \Delta\mathbf{r}_{j'}$$

Tomando ahora promedios en el radio de giro:

$$\langle R_g^2 \rangle = \frac{1}{N}\sum \langle \mathbf{r}_i^2 \rangle - \langle \mathbf{R}_{cm}^2 \rangle \tag{2}$$

de forma que:

$$\langle \mathbf{r}_i^2 \rangle = \sum_{j,j'=1}^{i,i} \langle \Delta\mathbf{r}_j \Delta\mathbf{r}_{j'} \rangle = \sum_{j,j'=1}^{i,i} \langle \Delta\mathbf{r}_j^2 \rangle \, \delta_{j,j'} \rightarrow \quad \text{por ser un polímero ideal}$$

$$\rightarrow \langle \mathbf{r}_i^2 \rangle = l_p^2 \sum_{j,j'=1}^{i,i} \delta_{j,j'} = l_p^2 \, i \tag{3}$$

$$\langle \mathbf{R}_{cm}^2 \rangle = \frac{1}{N^2}\sum_{i,i'=1}^{N,N} \sum_{j,j'=1}^{i,i'} \langle \Delta\mathbf{r}_j \Delta\mathbf{r}_{j'} \rangle = \frac{l_p^2}{N^2}\sum_{i,i'=1}^{N,N} \text{mín}(i,\, i') \rightarrow$$

$$\rightarrow \langle \mathbf{R}_{cm}^2 \rangle = l_p^2 \left(\frac{N}{3} + \frac{1}{2} + \frac{1}{6N} \right) \tag{4}$$

Substituyendo ahora las expresiones 3 y 4 en 2 :

$$\langle R_g^2 \rangle = \frac{l_p^2}{N}\sum_i^N i - l_p^2 \left(\frac{N}{3} + \frac{1}{2} + \frac{1}{6N} \right) = \frac{l_p^2}{6}\left[N - \frac{1}{N} \right]$$

$$\langle R_g^2 \rangle \simeq \frac{l_p^2}{6} N$$

Que expresado en función de los parámetros del modelo queda:

$$\langle R_g^2 \rangle \simeq l_p \frac{L}{6} = b_0 \frac{k_B T}{k_\theta} \frac{L}{6}$$

Luego finalmente el radio de giro:

$$\left[\!\left[\sqrt{\langle R_g^2 \rangle} = \sqrt{b_0 \frac{L}{6} \frac{k_B T}{k_\theta}} \right]\!\right]$$

2. Códigos

Programas escritos en lenguaje C, utilizados para la realización de este trabajo.

1. ***Script*** este programa crea una configuración inicial, un fichero de parámetros del experimento de traslocación (incluyendo especificaciones sobre la cantidad de estadística y el número de experimentos), y lanza/gestiona el proceso *mdrun* del paquete *Gromacs.*

2. ***Elasticidad*** programa de análisis estructural del poro; extrae las frecuencias características para una serie de modos normales en el mapa k_b, k_θ

(1)

```
#include "LS_Funciones.h"

int main()
{
    int i,j,k,l;
    int index_simulacion;
    int index_estad;
    int pinoffset;
    int ntmpi,ntomp;
    int pid_hijo0;
    int pid_hijo1;
    int pid_padre;

    struct CANAL_struct *CANAL_punt=&CANAL;
    struct POL_struct *POL_punt=&POL;
    struct MEMBRANA_struct *MEMBRANA_punt=&MEMBRANA;
    struct VARIABLE_struct *VARIABLE_punt=&VARIABLE;


    printf("\n\nScript: LanzarSimulacion, TFM GDS, Unizar.\n\n");

    Genconf();

    LeerInput(CANAL_punt,POL_punt,MEMBRANA_punt,VARIABLE_punt);


    for(i=0;i<VARIABLE_punt->N_simulaciones;i++)
    {
        for(j=0;j<VARIABLE_punt->N_estadistica;j++)
            GeneraMDP(CANAL_punt,POL_punt,VARIABLE_punt,i,j);
    }



    VARIABLE_punt->MatrizSeguridad=(int**)malloc(sizeof(int*)*VARIABLE_punt->N_simulaciones);
    for(i=0;i<VARIABLE_punt->N_simulaciones;i++)
        VARIABLE_punt->MatrizSeguridad[i]=(int*)malloc(sizeof(int)*VARIABLE_punt->N_estadistica);

    VARIABLE_punt->ArraySeguridad=(int*)malloc(sizeof(int)*VARIABLE_punt->N_simulaciones);



    for(i=0;i<VARIABLE_punt->N_simulaciones;i++)
    {
        for(j=0;j<VARIABLE_punt->N_estadistica;j++)
            VARIABLE_punt->MatrizSeguridad[i][j]=0;
    }

    for(i=0;i<VARIABLE_punt->N_simulaciones;i++)
        VARIABLE_punt->ArraySeguridad[i]=0;




    for(j=0;j<10;j++)
    {
        pid_padre=0;
        pid_hijo0=-1;
        pid_hijo1=-1;

        if(VARIABLE_punt->N_simulaciones==1)
        {
            ntmpi=4;

            pid_hijo0=fork();
            if(pid_hijo0==0)
            {
```

```
            pid_padre=-1;
            pid_hijo1=-1;
            index_simulacion=0;
        }
    }

    if(VARIABLE_punt->N_simulaciones>1)
    {
        ntmpi=2;

        pid_hijo0=fork();
        if(pid_hijo0==0)
        {
            pid_padre=-1;
            pid_hijo1=-1;
            index_simulacion=0;
        }
        if(pid_padre==0)
            pid_hijo1=fork();
        if(pid_hijo1==0)
        {
            pid_hijo0=-1;
            pid_padre=-1;
            index_simulacion=1;
        }
    }


    if(pid_padre==-1)
    {

        for(index_simulacion=index_simulacion;index_simulacion<VARIABLE_punt->N_simulaciones;
             index_simulacion=index_simulacion+2)
        {
            if(index_simulacion%2==0)
                pinoffset=0;
            else
                pinoffset=2;


            if(VARIABLE_punt->ArraySeguridad[index_simulacion]==0)
            {
                GenerarEstructuras(CANAL_punt,POL_punt,MEMBRANA_punt,VARIABLE_punt,
                    index_simulacion);

                if(strcmp(VARIABLE_punt->conf_externa,"si")==0)
                    UtilizaEstructuraPredefinida(CANAL_punt,POL_punt,MEMBRANA_punt,VARIABLE_punt
                        ,index_simulacion);


                Prepara_FicherosParaTermalizacion(VARIABLE_punt,CANAL_punt,POL_punt,
                    MEMBRANA_punt,pinoffset,ntmpi,ntomp,index_simulacion);


                Simula_Termalizacion(CANAL_punt,POL_punt,MEMBRANA_punt,VARIABLE_punt,pinoffset,
                    ntmpi,ntomp,index_simulacion);


                Prepara_FicherosParaEquilibrado(VARIABLE_punt,CANAL_punt,POL_punt,MEMBRANA_punt,
                    pinoffset,ntmpi,ntomp,index_simulacion);


                Simula_Equilibrado(CANAL_punt,POL_punt,MEMBRANA_punt,VARIABLE_punt,pinoffset,
                    ntmpi,ntomp,index_simulacion);



                for(index_estad=0;index_estad<VARIABLE_punt->N_estadistica;index_estad++)
                {


                    if(VARIABLE_punt->MatrizSeguridad[index_simulacion][index_estad]==0)
                    {

                        Prepara_FicherosParaTraslocacion(CANAL_punt,POL_punt,MEMBRANA_punt,
                            VARIABLE_punt,index_simulacion,index_estad);


                        Simula_Traslocacion(CANAL_punt,POL_punt,MEMBRANA_punt,VARIABLE_punt,
                            pinoffset,ntmpi,ntomp,index_simulacion,index_estad);


                        Prepara_FicherosParaAnalisis(VARIABLE_punt,CANAL_punt,POL_punt,
                            MEMBRANA_punt,index_simulacion,index_estad);
                    }

                }


                LimpiaEstructurasUsadas(CANAL_punt,POL_punt,MEMBRANA_punt,VARIABLE_punt,
                    index_simulacion);
            }
```

```
            }


            LiberarMemoriaReservada(CANAL_punt,POL_punt,MEMBRANA_punt,VARIABLE_punt); //Aqui libero
                 la memoria retenida por los hijos pero NO por el padre!
            exit(0); //fin de los hijos: ya estan hechas todas las simulaciones
        }

        wait(NULL);
        wait(NULL);

        if(GarantizoQueTodoEstaBien(CANAL_punt,POL_punt,MEMBRANA_punt,VARIABLE_punt)==1)
            break;
    }

    if(strcmp(VARIABLE_punt->pulling,"si")==0)
        AnalisisDeResultados(CANAL_punt,POL_punt,MEMBRANA_punt,VARIABLE_punt);

    LiberarMemoriaReservada(CANAL_punt,POL_punt,MEMBRANA_punt,VARIABLE_punt);


    ///FIN DEL PROGRAMA
    return 0;
}
```

1.1) Librería del programa *script*

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>

#define IA 16807
#define IM 2147483647
#define AM (1.0/IM)
#define IQ 127773
#define IR 2836
#define NTAB 32
#define NDIV (1+(IM-1)/NTAB)
#define EPS 1.2e-7
#define RNMX (1.0-EPS)

#define PI 3.1415926536
#define kB 1.3806488e-26 //kJ/K
#define Navogadro 6.02214179e+23 //mol^-1


struct CANAL_struct
{
    int sel;
    float radio;
    float longitudInterior;
    float longitudExterior;
    float *bond_fuerza;
    float bond_reposo;
    float *bending_fuerza;
    float bending_ang;
    float phi;
    int n;
    int M;
    int Nb;
    int NExterior;
    char Borde_poro[256];
    char Cuerpo_poro[256];
} CANAL;

struct POL_struct
{
    int sel;
    float bond_fuerza;
    float bond_reposo;
    float bending_fuerza;
    float bending_ang;
    float posicion;
    float *longitud;
    int n;
    char Borde_polimero[256];
    char Cuerpo_polimero[256];
} POL;

struct MEMBRANA_struct
{
    int sel;
    float area;
    float lado;
    int NumeroDSPC;
} MEMBRANA;
```

```
struct VARIABLE_struct
{
    int SISTEMA_n;
    int N_simulaciones;
    int N_estadistica;
    int AGUA_sel;
    float term_tolerancia;
    float term_tol_derivada;
    int N_term;
    int N_Lterm;
    int N_sim;
    float N_sim_cte;
    float tau_t;
    float ref_t;
    float *pull_rate;
    float *pull_fuerza;
    float pull_k;
    float radio_giro;
    float tiempo_extra;
    char pulling[256];
    char PintaTrayectoria[256];
    char Acoplo_presion[256];
    char conf_externa[256];
    char pulling_modo[256];
    int **MatrizSeguridad;
    int *ArraySeguridad;
    char campo_E[256];
    float *Ezfrecuencia;
    float Ezintensidad;
    float Exintensidad;
    float Eyintensidad;
} VARIABLE;


int Igual(float a, float b);

void SumaVector(float *a, float *b, float *result);

void RestaVector(float *a, float *b, float *result);

float ProductoEscalar(float *a, float *b);

float AnguloEntreVectores(float *a, float *b);

void ProductoVectorial(float *a, float *b, float *result);

void AsignaVector(float *a, float *result);

void Rotacion(float *EjeRotacion, float *Vector, float angulo);

float Distancia(float **Rx,float **Ry, float **Rz,int m1, int n1,int m2, int n2);

void MedVar(float *distribucion, int N, float* media, float* desviacion);

float ran1(long *idum);

void GenerarEstructuras(struct CANAL_struct* CANAL_punt,struct POL_struct* POL_punt, struct
    MEMBRANA_struct* MEMBRANA_punt,struct VARIABLE_struct* VARIABLE_punt,int index_simulacion);

void ConstruyeCanal(float CANAL_radio, float CANAL_bond_reposo, float CANAL_bond_fuerza, float
    CANAL_bending_ang, float CANAL_bending_fuerza, int CANAL_Nb,int CANAL_NExterior, int CANAL_M,
    float CANAL_phi, float **CANAL_X, float **CANAL_V, int *CANAL_residuo, int *CANAL_chgrp, float
    *CANAL_box, float *CANAL_Angle, int index_simulacion,char* Borde_poro,char* Cuerpo_poro);

void ConstruyeDisco(float **Rx,float **Ry,float **Rz, float CANAL_bond_reposo,float CANAL_radio,
    float phi, int M, int i);

void MatrizDistancia(float **MD,float **Rx,float **Ry, float **Rz,int M, int Nb);

void MatrizVecinos(int **MV,int M, int Nb,float **MatrizDistancia,float CANAL_bond_reposo);

void GeneraBond(int **MatrizEnlaces,int **MVecinos, int M, int Nb);

void GeneraAngle(int ***MatrizAngulos,int **MVecinos, int M, int Nb);

void ConstruyeTopol_canal(float CANAL_bond_reposo,float CANAL_bond_fuerza, float CANAL_bending_ang,
    float CANAL_bending_fuerza,int M, int Nb,int NExterior,int **MEnlaces, int ***MAngulos, int
    index_simulacion, char* Borde_poro, char* Cuerpo_poro);

void ConstruyePolimero(int POL_n,float POL_bond_reposo, float POL_bond_fuerza,float POL_bending_ang,
    float POL_bending_fuerza,float **POL_X, float **POL_V, int *POL_residuo, int *POL_chgrp, float
    *POL_box, float *POL_Angle, int index_simulacion ,char* Borde_polimero, char* Cuerpo_polimero);

void ConstruyeTopol_polimero(int POL_n,float POL_bond_reposo,float POL_bond_fuerza,float
    POL_bending_ang,float POL_bending_fuerza,int index_simulacion,char* Borde_polimero,char*
    Cuerpo_polimero);

void ConstruyeDSPC(float **DSPC_X, float **DSPC_V, int *DSPC_residuo, int *DSPC_chgrp, float *
    DSPC_box, float *DSPC_Angle, char *DSPC_atomname, float angulo,int AGUA_sel ,int
    index_simulacion, char* campoE);

void UbicarEnMembrana(float ****MEMBRANA_X, float **DSPC_X, int DSPC_n, int i, int j);
```

```
int AsignaOrientacion_Membrana(int i, int j, int MEMBRANA_n);

int AhuecarMembrana(float ****MEMBRANA_X,int MEMBRANA_n,float radio);

void ConstruyeTopol_DSPC(int index_simulacion, int AGUA_sel,char* campoE);

void MoverEstructura(float **estructura,int N,float theta, float phi, float Rx, float Ry, float Rz,
     long idum, float *Angle);

void Calcula_CM(float **estructura, int N, float *CentroMasa);

void ConstruyeIndices(int SISTEMA_n, int CANAL_n,int CANAL_M,int CANAL_Nb,float CANAL_bond_reposo,
     int POL_n,int DSPC_n ,int NumeroDSPC, int index_simulacion,int CANAL_sel,int POL_sel, float
     POL_posicion,float POL_bond_reposo, int MEMBRANA_sel);

void EscribeFicheroTopol(int NumeroDSPC, int NumeroW, int index_simulacion, int CANAL_sel, int
     POL_sel, int MEMBRANA_sel);

void Genconf();

float CalculaRadioGiro(int index_simulacion, char* Carpeta, char* GROfile);

float TiempoTraslocacion(struct CANAL_struct* CANAL_punt,struct POL_struct* POL_punt, struct
     MEMBRANA_struct* MEMBRANA_punt,struct VARIABLE_struct* VARIABLE_punt,int index_simulacion, int
     index_estad);

void AnalisisDeResultados(struct CANAL_struct* CANAL_punt,struct POL_struct* POL_punt, struct
     MEMBRANA_struct* MEMBRANA_punt,struct VARIABLE_struct* VARIABLE_punt);

void LeerInput(struct CANAL_struct* CANAL_punt, struct POL_struct* POL_punt, struct MEMBRANA_struct*
      MEMBRANA_punt, struct VARIABLE_struct* VARIABLE_punt);

void LiberarMemoriaReservada(struct CANAL_struct* CANAL_punt,struct POL_struct* POL_punt, struct
     MEMBRANA_struct* MEMBRANA_punt,struct VARIABLE_struct* VARIABLE_punt);
void LimpiaEstructurasUsadas(struct CANAL_struct* CANAL_punt,struct POL_struct* POL_punt, struct
     MEMBRANA_struct* MEMBRANA_punt,struct VARIABLE_struct* VARIABLE_punt,int index_simulacion);


void GeneraMDP(struct CANAL_struct* CANAL_punt,struct POL_struct* POL_punt,struct VARIABLE_struct*
     VARIABLE_punt, int index_simulacion,int index_estad);

void Prepara_FicherosParaTermalizacion(struct VARIABLE_struct* VARIABLE_punt,struct CANAL_struct*
     CANAL_punt,struct POL_struct* POL_punt,struct MEMBRANA_struct* MEMBRANA_punt,int pinoffset,int
     ntmpi, int ntomp, int index_simulacion);

void Simula_Termalizacion(struct CANAL_struct* CANAL_punt,struct POL_struct* POL_punt, struct
     MEMBRANA_struct* MEMBRANA_punt,struct VARIABLE_struct* VARIABLE_punt,int pinoffset,int ntmpi,
     int ntomp,int index_simulacion);

void Prepara_FicherosParaEquilibrado(struct VARIABLE_struct* VARIABLE_punt,struct CANAL_struct*
     CANAL_punt,struct POL_struct* POL_punt,struct MEMBRANA_struct* MEMBRANA_punt,int pinoffset,int
     ntmpi, int ntomp,int index_simulacion);

void Simula_Equilibrado(struct CANAL_struct* CANAL_punt,struct POL_struct* POL_punt, struct
     MEMBRANA_struct* MEMBRANA_punt,struct VARIABLE_struct* VARIABLE_punt,int pinoffset,int ntmpi,
     int ntomp, int index_simulacion);

void Prepara_FicherosParaTraslocacion(struct CANAL_struct* CANAL_punt,struct POL_struct* POL_punt,
     struct MEMBRANA_struct* MEMBRANA_punt,struct VARIABLE_struct* VARIABLE_punt,int
     index_simulacion, int index_estad);

void Simula_Traslocacion(struct CANAL_struct* CANAL_punt,struct POL_struct* POL_punt, struct
     MEMBRANA_struct* MEMBRANA_punt,struct VARIABLE_struct* VARIABLE_punt,int pinoffset,int ntmpi,
     int ntomp,int index_simulacion,int index_estad);

void Prepara_FicherosParaAnalisis(struct VARIABLE_struct* VARIABLE_punt,struct CANAL_struct*
     CANAL_punt,struct POL_struct* POL_punt,struct MEMBRANA_struct* MEMBRANA_punt,int
     index_simulacion,int index_estad);

void UtilizaEstructuraPredefinida(struct CANAL_struct* CANAL_punt,struct POL_struct* POL_punt,
     struct MEMBRANA_struct* MEMBRANA_punt,struct VARIABLE_struct* VARIABLE_punt, int
     index_simulacion);

int GarantizoQueTodoEstaBien(struct CANAL_struct* CANAL_punt,struct POL_struct* POL_punt, struct
     MEMBRANA_struct* MEMBRANA_punt,struct VARIABLE_struct* VARIABLE_punt);




/*************************
 *                       *
 * FUNCIONES GENERALES   *
 *                       *
 *************************/
///FUNCIONES OPERATIVAS/////////////////////////////////////////////////////////////////////
int Igual(float a, float b){   //Devuelve 1 si son iguales, 0 si son diferentes
    float epsilon=0.0000001;

    if (fabs(a-b)<epsilon)
        return 1;

    else
```

```
        return 0;
}


void SumaVector(float *a, float *b, float *result){
    int i;
    for(i=0;i<3;i++)
        result[i]=a[i]+b[i];
}

void RestaVector(float *a, float *b, float *result){
    int i;
    for(i=0;i<3;i++)
        result[i]=a[i]-b[i];
}

float ProductoEscalar(float *a, float *b){
    float result=0;
    int i;
    for(i=0;i<3;i++){
        result+=a[i]*b[i];
    }

    return result;
}

float AnguloEntreVectores(float *a, float *b){
    int i;
    float mod_a,mod_b,angulo;
    mod_a=sqrt(ProductoEscalar(a,a));
    mod_b=sqrt(ProductoEscalar(b,b));

    angulo=acos(ProductoEscalar(a,b)/(mod_a*mod_b));
    return angulo;
}

void ProductoVectorial(float *a, float *b, float *result){

    result[0]=a[1]*b[2]-a[2]*b[1];
    result[1]=a[2]*b[0]-a[0]*b[2];
    result[2]=a[0]*b[1]-a[1]*b[0];

}

void AsignaVector(float *a, float *result){
    result[0]=a[0];
    result[1]=a[1];
    result[2]=a[2];
}

void Rotacion(float *EjeRotacion, float *Vector, float angulo){
    int i,j;
    float MatrizRotacion[3][3];
    float Vector_auxiliar[3]={0,0,0};
    float mod_Eje;

    angulo=angulo*PI/180; ///Convertir angulos a radianes (para operar con math.h)
    mod_Eje=sqrt(ProductoEscalar(EjeRotacion, EjeRotacion)); ///Modulo del eje de rotacion. Debe ser
        1, pero por si acaso lo normalizo antes de seguir.

    EjeRotacion[0]=EjeRotacion[0]/mod_Eje;
    EjeRotacion[1]=EjeRotacion[1]/mod_Eje;
    EjeRotacion[2]=EjeRotacion[2]/mod_Eje;

    MatrizRotacion[0][0]=cos(angulo)+EjeRotacion[0]*EjeRotacion[0]*(1-cos(angulo));
    MatrizRotacion[0][1]=EjeRotacion[0]*EjeRotacion[1]*(1-cos(angulo))-EjeRotacion[2]*sin(angulo);
    MatrizRotacion[0][2]=EjeRotacion[0]*EjeRotacion[2]*(1-cos(angulo))+EjeRotacion[1]*sin(angulo);
    MatrizRotacion[1][0]=EjeRotacion[0]*EjeRotacion[1]*(1-cos(angulo))+EjeRotacion[2]*sin(angulo);
    MatrizRotacion[1][1]=cos(angulo)+EjeRotacion[1]*EjeRotacion[1]*(1-cos(angulo));
    MatrizRotacion[1][2]=EjeRotacion[1]*EjeRotacion[2]*(1-cos(angulo))-EjeRotacion[0]*sin(angulo);
    MatrizRotacion[2][0]=EjeRotacion[0]*EjeRotacion[2]*(1-cos(angulo))-EjeRotacion[1]*sin(angulo);
    MatrizRotacion[2][1]=EjeRotacion[1]*EjeRotacion[2]*(1-cos(angulo))+EjeRotacion[0]*sin(angulo);
    MatrizRotacion[2][2]=cos(angulo)+EjeRotacion[2]*EjeRotacion[2]*(1-cos(angulo)); ///Matriz de
        rotacion construida.

    for(i=0;i<3;i++){
        for(j=0;j<3;j++)
            Vector_auxiliar[i]+=MatrizRotacion[i][j]*Vector[j];
    }

    Vector[0]=Vector_auxiliar[0];
    Vector[1]=Vector_auxiliar[1];
    Vector[2]=Vector_auxiliar[2];

}

float Distancia(float **Rx,float **Ry, float **Rz,int m1, int n1,int m2, int n2){
    return(sqrt((Rx[m1][n1]-Rx[m2][n2])*(Rx[m1][n1]-Rx[m2][n2])+(Ry[m1][n1]-Ry[m2][n2])*(Ry[m1][n1]-
        Ry[m2][n2])+(Rz[m1][n1]-Rz[m2][n2])*(Rz[m1][n1]-Rz[m2][n2])));
}

void MedVar(float *distribucion, int N, float* media, float* desviacion)
{
```

```
    int i;
    float promedio=0;
    float varianza=0;


    for(i=0;i<N;i++)
        promedio+=distribucion[i];

    promedio=promedio/N;

    for(i=0;i<N;i++)
    {
        varianza+=(distribucion[i]-promedio)*(distribucion[i]-promedio);
    }
    varianza=varianza/(N-1);

    *media=promedio;
    *desviacion=sqrt(varianza);
}

float ran1(long *idum){
    int j;
    long k;
    static long iy=0;
    static long iv[NTAB];
    float temp;
    if (*idum <= 0 || !iy) {
        if (-(*idum) < 1) *idum=1;
        else *idum = -(*idum);

        for (j=NTAB+7;j>=0;j--) {
            k=(*idum)/IQ;
            *idum=IA*(*idum-k*IQ)-IR*k;
            if (*idum < 0)
                *idum += IM;
            if (j < NTAB)
                iv[j] = *idum;
        }
        iy=iv[0];
    }
    k=(*idum)/IQ;
    *idum=IA*(*idum-k*IQ)-IR*k;
    if (*idum < 0) *idum += IM;
    j=iy/NDIV;
    iy=iv[j];
    iv[j] = *idum;
    if ((temp=AM*iy) > RNMX) return RNMX;
    else return temp;
}


///CONSTRUCCION DE ESTRUCTURAS//////////////////////////////////////
void GenerarEstructuras(struct CANAL_struct* CANAL_punt,struct POL_struct* POL_punt, struct
    MEMBRANA_struct* MEMBRANA_punt,struct VARIABLE_struct* VARIABLE_punt,int index_simulacion)
{
    int i,j,k;
    char comando[8192];

    char CANAL_name[256];
    int *CANAL_residuo;
    int *CANAL_chgrp;
    int *CANAL_index;
    float **CANAL_X;
    float **CANAL_V;
    float CANAL_box[3];
    float CANAL_Angle[2];

    char POL_name[256];
    int *POL_residuo;
    int *POL_chgrp;
    int *POL_index;
    float **POL_X;
    float **POL_V;
    float POL_box[3];
    float POL_Angle[2];

    int DSPC_n=14;
    int *DSPC_residuo;
    char *DSPC_atomname;
    int *DSPC_chgrp;
    float **DSPC_X;
    float **DSPC_V;
    float DSPC_box[3];
    float DSPC_Angle[2];
    int MEMBRANA_n;
    int eliminados;
    int ***MEMBRANA_index_a;
    int ***MEMBRANA_index_b;
    float ****MEMBRANA_X_a;
    float ****MEMBRANA_X_b;

    int AtomNumber=0;
    char Configuracion_name[256];
```

```
long idum=-(long)time(NULL);
FILE* SISTEMA_nuevo;
FILE *pipe;


sprintf(Configuracion_name,"SISTEMA_conf_%d.gro",index_simulacion);
SISTEMA_nuevo=fopen(Configuracion_name,"w");


///-----------------CANAL-----------------------------------------------------

if(CANAL_punt->sel==0)
    CANAL_punt->n=1;


CANAL_residuo=(int*)malloc(sizeof(int)*CANAL_punt->n);


CANAL_chgrp=(int*)malloc(sizeof(int)*CANAL_punt->n);
CANAL_index=(int*)malloc(sizeof(int)*CANAL_punt->n);

CANAL_X=(float**)malloc(sizeof(float*)*CANAL_punt->n);
for(i=0;i<CANAL_punt->n;i++)
    *(CANAL_X+i)=(float*)malloc(sizeof(float)*3);

CANAL_V=(float**)malloc(sizeof(float*)*CANAL_punt->n);
for(i=0;i<CANAL_punt->n;i++)
    *(CANAL_V+i)=(float*)malloc(sizeof(float)*3);


if(CANAL_punt->sel==1){
    printf("\n\nPORO\n----------------\n");

    ConstruyeCanal(CANAL_punt->radio,CANAL_punt->bond_reposo,CANAL_punt->bond_fuerza[
        index_simulacion],CANAL_punt->bending_ang,CANAL_punt->bending_fuerza[index_simulacion],
        CANAL_punt->Nb,CANAL_punt->NExterior,CANAL_punt->M,CANAL_punt->phi,CANAL_X,CANAL_V,
        CANAL_residuo,CANAL_chgrp,CANAL_box,CANAL_Angle,index_simulacion,CANAL_punt->Borde_poro
        ,CANAL_punt->Cuerpo_poro);

    MoverEstructura(CANAL_X,CANAL_punt->n,0, 0, -0.4, -0.4, 0, idum,CANAL_Angle);
}

for(i=0;i<CANAL_punt->n;i++){
    AtomNumber++;
    CANAL_index[i]=AtomNumber;
}



///-----------------POLY-----------------------------------------------------
POL_punt->n=(int)(POL_punt->longitud[index_simulacion]/POL_punt->bond_reposo);

if(POL_punt->sel==0)
    POL_punt->n=1;

POL_residuo=(int*)malloc(sizeof(int)*POL_punt->n);


POL_chgrp=(int*)malloc(sizeof(int)*POL_punt->n);
POL_index=(int*)malloc(sizeof(int)*POL_punt->n);

POL_X=(float**)malloc(sizeof(float*)*(POL_punt->n));
for(i=0;i<(POL_punt->n);i++)
    *(POL_X+i)=(float*)malloc(sizeof(float)*3);

POL_V=(float**)malloc(sizeof(float*)*(POL_punt->n));
for(i=0;i<(POL_punt->n);i++)
    *(POL_V+i)=(float*)malloc(sizeof(float)*3);



if(POL_punt->sel==1){
    printf("\n\nPOLIMERO\n----------------\n");

    ConstruyePolimero(POL_punt->n,POL_punt->bond_reposo,POL_punt->bond_fuerza,POL_punt->
        bending_ang,POL_punt->bending_fuerza,POL_X,POL_V,POL_residuo,POL_chgrp,POL_box,
        POL_Angle,index_simulacion,POL_punt->Borde_polimero,POL_punt->Cuerpo_polimero);

    MoverEstructura(POL_X,POL_punt->n,0, 0, -0.5, -0.5,POL_punt->posicion*POL_punt->bond_reposo*
        POL_punt->n*0.5, idum,POL_Angle);
}

for(i=0;i<POL_punt->n;i++){
    AtomNumber++;
    POL_index[i]=AtomNumber;

}



///-----------------MEMBRANA-----------------------------------------------------
```

```
if(MEMBRANA_punt->sel==0)
      DSPC_n=1;


MEMBRANA_punt->lado=sqrt(MEMBRANA_punt->area);
MEMBRANA_n=(int)MEMBRANA_punt->lado;

DSPC_residuo=(int*)malloc(sizeof(int)*DSPC_n);

DSPC_atomname=(char*)malloc(sizeof(char)*DSPC_n);

DSPC_chgrp=(int*)malloc(sizeof(int)*DSPC_n);

DSPC_X=(float**)malloc(sizeof(float*)*DSPC_n);
for(i=0;i<DSPC_n;i++)
      *(DSPC_X+i)=(float*)malloc(sizeof(float)*3);

DSPC_V=(float**)malloc(sizeof(float*)*DSPC_n);
for(i=0;i<DSPC_n;i++)
      *(DSPC_V+i)=(float*)malloc(sizeof(float)*3);

MEMBRANA_index_a=(int***)malloc(sizeof(int**)*MEMBRANA_n);
for(i=0;i<MEMBRANA_n;i++)
      MEMBRANA_index_a[i]=(int**)malloc(sizeof(int*)*MEMBRANA_n);
for(i=0;i<MEMBRANA_n;i++){
      for(j=0;j<MEMBRANA_n;j++){
            MEMBRANA_index_a[i][j]=(int*)malloc(sizeof(int)*14);
      }
}

MEMBRANA_index_b=(int***)malloc(sizeof(int**)*MEMBRANA_n);
for(i=0;i<MEMBRANA_n;i++)
      MEMBRANA_index_b[i]=(int**)malloc(sizeof(int*)*MEMBRANA_n);
for(i=0;i<MEMBRANA_n;i++){
      for(j=0;j<MEMBRANA_n;j++){
            MEMBRANA_index_b[i][j]=(int*)malloc(sizeof(int)*14);
      }
}



MEMBRANA_X_a=(float****)malloc(sizeof(float***)*MEMBRANA_n);

for(i=0;i<MEMBRANA_n;i++)
      MEMBRANA_X_a[i]=(float***)malloc(sizeof(float**)*MEMBRANA_n);

for(i=0;i<MEMBRANA_n;i++){
      for(j=0;j<MEMBRANA_n;j++)
            MEMBRANA_X_a[i][j]=(float**)malloc(sizeof(float*)*DSPC_n);
}
for(i=0;i<MEMBRANA_n;i++){
      for(j=0;j<MEMBRANA_n;j++){
            for(k=0;k<DSPC_n;k++)
                  MEMBRANA_X_a[i][j][k]=(float*)malloc(sizeof(float)*3);
      }
}

MEMBRANA_X_b=(float****)malloc(sizeof(float***)*MEMBRANA_n);

for(i=0;i<MEMBRANA_n;i++)
      MEMBRANA_X_b[i]=(float***)malloc(sizeof(float**)*MEMBRANA_n);

for(i=0;i<MEMBRANA_n;i++){
      for(j=0;j<MEMBRANA_n;j++)
            MEMBRANA_X_b[i][j]=(float**)malloc(sizeof(float*)*DSPC_n);
}
for(i=0;i<MEMBRANA_n;i++){
      for(j=0;j<MEMBRANA_n;j++){
            for(k=0;k<DSPC_n;k++)
                  MEMBRANA_X_b[i][j][k]=(float*)malloc(sizeof(float)*3);
      }
}



if(MEMBRANA_punt->sel==1){
      printf("\n\nMEMBRANA\n----------\n");

      ConstruyeDSPC(DSPC_X,DSPC_V,DSPC_residuo,DSPC_chgrp,DSPC_box,DSPC_Angle,DSPC_atomname, 20,
            VARIABLE_punt->AGUA_sel,index_simulacion,VARIABLE_punt->campo_E);

      ///RELLENAR MEMBRANA_a:
      for(i=0;i<MEMBRANA_n;i++){
            for(j=0;j<MEMBRANA_n;j++){
                  MoverEstructura(DSPC_X,DSPC_n,0,180*ran1(&idum)-90,0.7*(i-0.5*MEMBRANA_n),0.7*(j
                        -0.5*MEMBRANA_n),1.6, idum,DSPC_Angle);
                  UbicarEnMembrana(MEMBRANA_X_a,DSPC_X,DSPC_n,i,j);
            }
      }


      ///RELLENAR MEMBRANA_b:
      for(i=0;i<MEMBRANA_n;i++){
```

```
        for(j=0;j<MEMBRANA_n;j++){
            MoverEstructura(DSPC_X,DSPC_n,180,180*ran1(&idum)-90,0.7*(i-0.5*MEMBRANA_n),0.7*(j
                -0.5*MEMBRANA_n),-1.6, idum,DSPC_Angle);
            UbicarEnMembrana(MEMBRANA_X_b,DSPC_X,DSPC_n,i,j);
        }
    }


    AhuecarMembrana(MEMBRANA_X_a,MEMBRANA_n,CANAL_punt->radio*1.6);
    eliminados=AhuecarMembrana(MEMBRANA_X_b,MEMBRANA_n,CANAL_punt->radio*1.6);

    for(i=0;i<MEMBRANA_n;i++){
        for(j=0;j<MEMBRANA_n;j++){
            for(k=0;k<DSPC_n;k++){
                if(MEMBRANA_X_a[i][j][0]!=NULL){
                    AtomNumber++;
                    MEMBRANA_index_a[i][j][k]=AtomNumber;
                }

            }
        }
    }



    for(i=0;i<MEMBRANA_n;i++){
        for(j=0;j<MEMBRANA_n;j++){
            for(k=0;k<DSPC_n;k++){
                if(MEMBRANA_X_b[i][j][0]!=NULL){
                    AtomNumber++;
                    MEMBRANA_index_b[i][j][k]=AtomNumber;
                }
            }
        }
    }

}



///ESCRITURA DEL SISTEMA:
MEMBRANA_punt->NumeroDSPC=2*MEMBRANA_punt->sel*(MEMBRANA_n*MEMBRANA_n-eliminados);
VARIABLE_punt->SISTEMA_n=CANAL_punt->sel*CANAL_punt->n+POL_punt->sel*POL_punt->n+MEMBRANA_punt->
    NumeroDSPC*DSPC_n;

fprintf(SISTEMA_nuevo,"%s\n","Sistema");

fprintf(SISTEMA_nuevo,"%d\n",VARIABLE_punt->SISTEMA_n);

if(CANAL_punt->sel==1){
    for(i=0;i<CANAL_punt->n;i++){
        if(i/CANAL_punt->M<CANAL_punt->NExterior || i/CANAL_punt->M+1>CANAL_punt->Nb-CANAL_punt
            ->NExterior)
            fprintf(SISTEMA_nuevo,"%5d %-5s %5c %5d %8.3f %8.3f %8.3f %8.3f %8.3f %8.3f\n",CANAL_residuo[
                i],"CHNNL",'G',CANAL_index[i],CANAL_X[i][0],CANAL_X[i][1],CANAL_X[i][2],CANAL_V
                [i][0],CANAL_V[i][1],CANAL_V[i][2]);
        else
            fprintf(SISTEMA_nuevo,"%5d %-5s %5c %5d %8.3f %8.3f %8.3f %8.3f %8.3f %8.3f\n",CANAL_residuo[
                i],"CHNNL",'C',CANAL_index[i],CANAL_X[i][0],CANAL_X[i][1],CANAL_X[i][2],CANAL_V
                [i][0],CANAL_V[i][1],CANAL_V[i][2]);
    }
}

if(POL_punt->sel==1){
    for(i=0;i<POL_punt->n;i++){
        if(i==0)
            fprintf(SISTEMA_nuevo,"%5d %-5s %5c %5d %8.3f %8.3f %8.3f %8.3f %8.3f %8.3f\n",1+CANAL_punt->
                sel*CANAL_residuo[CANAL_punt->n-1],"PEG",'H',POL_index[i],POL_X[i][0],POL_X[i
                ][1],POL_X[i][2],POL_V[i][0],POL_V[i][1],POL_V[i][2]);
        else
            fprintf(SISTEMA_nuevo,"%5d %-5s %5c %5d %8.3f %8.3f %8.3f %8.3f %8.3f %8.3f\n",1+CANAL_punt->
                sel*CANAL_residuo[CANAL_punt->n-1],"PEG",'C',POL_index[i],POL_X[i][0],POL_X[i
                ][1],POL_X[i][2],POL_V[i][0],POL_V[i][1],POL_V[i][2]);
    }
}

if(MEMBRANA_punt->sel==1){
    for(j=0;j<MEMBRANA_n;j++){
        for(k=0;k<MEMBRANA_n;k++){
            if(MEMBRANA_X_a[j][k][0]!=NULL){
                for(i=0;i<DSPC_n;i++)
                    fprintf(SISTEMA_nuevo,"%5d %-5s %5c %5d %8.3f %8.3f %8.3f %8.3f %8.3f %8.3f\n",j*
                        MEMBRANA_n+k+1+POL_punt->sel*POL_residuo[POL_punt->n-1]+CANAL_punt->sel
                        *CANAL_residuo[CANAL_punt->n-1],"DSPC",DSPC_atomname[i],
                        MEMBRANA_index_a[j][k][i],MEMBRANA_X_a[j][k][i][0],MEMBRANA_X_a[j][k][i
                        ][1],MEMBRANA_X_a[j][k][i][2],DSPC_V[i][0],DSPC_V[i][1],DSPC_V[i][2]);
            }
        }
    }

    for(j=0;j<MEMBRANA_n;j++){
        for(k=0;k<MEMBRANA_n;k++){
```

```
                if(MEMBRANA_X_b[j][k][0]!=NULL){
                    for(i=0;i<DSPC_n;i++)
                        fprintf(SISTEMA_nuevo," %5d %-5s %5c %5d %8.3f %8.3f %8.3f %8.3f %8.3f %8.3f\n",
                            MEMBRANA_n*MEMBRANA_n+j*MEMBRANA_n+k+1+POL_punt->sel*POL_residuo[
                            POL_punt->n-1]+CANAL_punt->sel*CANAL_residuo[CANAL_punt->n-1],"DSPC",
                            DSPC_atomname[i],MEMBRANA_index_b[j][k][i],MEMBRANA_X_b[j][k][i][0],
                            MEMBRANA_X_b[j][k][i][1],MEMBRANA_X_b[j][k][i][2],DSPC_V[i][0],DSPC_V[i
                            ][1],DSPC_V[i][2]);
                }
            }
        }
    }

    fprintf(SISTEMA_nuevo," %f\t %f\t %f\n",POL_box[0]+CANAL_box[0]+MEMBRANA_n*DSPC_box[0],POL_box[1]+
        CANAL_box[1]+MEMBRANA_n*DSPC_box[1],POL_box[2]+CANAL_box[2]+MEMBRANA_n*DSPC_box[2]);



    ///ESCRITURA DE INDICES
    ConstruyeIndices(VARIABLE_punt->SISTEMA_n,CANAL_punt->n,CANAL_punt->M,CANAL_punt->Nb,CANAL_punt
        ->bond_reposo, POL_punt->n,DSPC_n,MEMBRANA_punt->NumeroDSPC,index_simulacion,CANAL_punt->
        sel,POL_punt->sel,POL_punt->posicion,POL_punt->bond_reposo,MEMBRANA_punt->sel);
    printf("\nNumero de DSPC total= %d\n",MEMBRANA_punt->NumeroDSPC);



    ///LIBERACION DE MEMORIA
    free(CANAL_residuo);

    free(CANAL_chgrp);
    free(CANAL_index);

    for(i=0;i<CANAL_punt->n;i++)
        free(*(CANAL_X+i));
    free(CANAL_X);

    for(i=0;i<CANAL_punt->n;i++)
        free(*(CANAL_V+i));
    free(CANAL_V);



    free(POL_residuo);

    free(POL_chgrp);
    free(POL_index);

    for(i=0;i<(POL_punt->n);i++)
        free(*(POL_X+i));
    free(POL_X);

    for(i=0;i<(POL_punt->n);i++)
        free(*(POL_V+i));
    free(POL_V);



    free(DSPC_residuo);

    free(DSPC_atomname);

    free(DSPC_chgrp);

    for(i=0;i<DSPC_n;i++)
        free(*(DSPC_X+i));
    free(DSPC_X);

    for(i=0;i<DSPC_n;i++)
        free(*(DSPC_V+i));
    free(DSPC_V);


    for(i=0;i<MEMBRANA_n;i++){
        for(j=0;j<MEMBRANA_n;j++){
            free(MEMBRANA_index_a[i][j]);
        }
    }
    for(i=0;i<MEMBRANA_n;i++)
        free(MEMBRANA_index_a[i]);
    free(MEMBRANA_index_a);


    for(i=0;i<MEMBRANA_n;i++){
        for(j=0;j<MEMBRANA_n;j++){
            free(MEMBRANA_index_b[i][j]);
        }
    }
    for(i=0;i<MEMBRANA_n;i++)
        free(MEMBRANA_index_b[i]);
    free(MEMBRANA_index_b);


    for(i=0;i<MEMBRANA_n;i++){
```

```
            for(j=0;j<MEMBRANA_n;j++){
                for(k=0;k<DSPC_n;k++)
                    free(MEMBRANA_X_a[i][j][k]);
            }
        }

        for(i=0;i<MEMBRANA_n;i++){
            for(j=0;j<MEMBRANA_n;j++)
                free(MEMBRANA_X_a[i][j]);
        }


        for(i=0;i<MEMBRANA_n;i++)
            free(MEMBRANA_X_a[i]);

        free(MEMBRANA_X_a);

        for(i=0;i<MEMBRANA_n;i++){
            for(j=0;j<MEMBRANA_n;j++){
                for(k=0;k<DSPC_n;k++)
                    free(MEMBRANA_X_b[i][j][k]);
            }
        }

        for(i=0;i<MEMBRANA_n;i++){
            for(j=0;j<MEMBRANA_n;j++)
                free(MEMBRANA_X_b[i][j]);
        }


        for(i=0;i<MEMBRANA_n;i++)
            free(MEMBRANA_X_b[i]);

        free(MEMBRANA_X_b);


        fflush(SISTEMA_nuevo);
        fclose(SISTEMA_nuevo);

    }


    ///Construccion del Canal

    void ConstruyeCanal(float CANAL_radio, float CANAL_bond_reposo, float CANAL_bond_fuerza, float
         CANAL_bending_ang, float CANAL_bending_fuerza, int CANAL_Nb, int CANAL_NExterior, int CANAL_M,
         float CANAL_phi, float **CANAL_X, float **CANAL_V, int *CANAL_residuo, int *CANAL_chgrp, float
         *CANAL_box, float *CANAL_Angle, int index_simulacion, char* Borde_poro, char* Cuerpo_poro){

        ///Definicion de variables auxiliares (para la programacion)
        int n,m,i,j;

        ///Definicion del contenedor de posiciones del sistema
        float **Rx;
        float **Ry;
        float **Rz;
        float **MDistancia;    //Matriz de distancias.
        int **MVecinos;
        int **MEnlaces;
        int ***MAngulos;


        ///Definicion de parametros del canal*****************
        printf("\nNOTA: se han introducido los parametros:\nCANAL_bond_reposo=%.2f\nCANAL_radio=%.1f\
             nLongitud=%d\n\n",CANAL_bond_reposo,CANAL_radio,CANAL_Nb);
        printf("Estos parametros suponen:\nAmplitud_angular=%.2f\tBeads/vuelta=%d\n\n",CANAL_phi,CANAL_M
             );
        CANAL_phi=2*PI/CANAL_M;
        CANAL_radio=CANAL_bond_reposo*sqrt(1/(2*(1-cos(CANAL_phi))));
        printf("Para ajustar el cilindro, se modifican los parametros:\nAmplitud angular'=%.2f\
             tCANAL_radio'=%.3f\n\n",CANAL_phi,CANAL_radio);



        ///Asigancion de memoria al sistema*****************
        Rx=(float**)malloc(sizeof(float*)*CANAL_M);
        Ry=(float**)malloc(sizeof(float*)*CANAL_M);
        Rz=(float**)malloc(sizeof(float*)*CANAL_M);
        for(i=0;i<CANAL_M;i++){
            *(Rx+i)=(float*)malloc(sizeof(float)*CANAL_Nb);
            *(Ry+i)=(float*)malloc(sizeof(float)*CANAL_Nb);
            *(Rz+i)=(float*)malloc(sizeof(float)*CANAL_Nb);
        }

        MDistancia=(float**)malloc(sizeof(float*)*CANAL_M*CANAL_Nb);
        for(i=0;i<(CANAL_M*CANAL_Nb);i++)
            *(MDistancia+i)=(float*)malloc(sizeof(float)*CANAL_M*CANAL_Nb);

        MVecinos=(int**)malloc(sizeof(int*)*CANAL_M*CANAL_Nb);
        for(i=0;i<CANAL_M*CANAL_Nb;i++)
            *(MVecinos+i)=(int*)malloc(sizeof(int)*CANAL_M*CANAL_Nb);

        MEnlaces=(int**)malloc(sizeof(int*)*CANAL_M*CANAL_Nb);
```

```
    for(i=0;i<(CANAL_M*CANAL_Nb);i++)
        *(MEnlaces+i)=(int*)malloc(sizeof(int)*CANAL_M*CANAL_Nb);


    MAngulos=(int***)malloc(sizeof(int**)*CANAL_M*CANAL_Nb);
    for(i=0;i<(CANAL_M*CANAL_Nb);i++)
        *(MAngulos+i)=(int**)malloc(sizeof(int*)*CANAL_M*CANAL_Nb);

    for(i=0;i<CANAL_M*CANAL_Nb;i++){
        for(j=0;j<CANAL_M*CANAL_Nb;j++)
            *(*(MAngulos+i)+j)=(int*)malloc(sizeof(int)*CANAL_M*CANAL_Nb);
    }



    ///Construccion del sistema
    for(i=0;i<CANAL_Nb;i++)
        ConstruyeDisco(Rx,Ry,Rz,CANAL_bond_reposo,CANAL_radio,CANAL_phi,CANAL_M,i); //Con este bucle
            rellenamos las matrices R

    MatrizDistancia(MDistancia,Rx,Ry,Rz,CANAL_M,CANAL_Nb);

    MatrizVecinos(MVecinos,CANAL_M,CANAL_Nb,MDistancia,CANAL_bond_reposo);

    GeneraBond(MEnlaces,MVecinos,CANAL_M,CANAL_Nb);
    GeneraAngle(MAngulos,MVecinos,CANAL_M,CANAL_Nb);
    ConstruyeTopol_canal(CANAL_bond_reposo,CANAL_bond_fuerza,CANAL_bending_ang,CANAL_bending_fuerza,
        CANAL_M,CANAL_Nb,CANAL_NExterior,MEnlaces,MAngulos,index_simulacion,Borde_poro,Cuerpo_poro)
        ;


    for(n=0;n<CANAL_Nb;n++){
        for(m=0;m<CANAL_M;m++){
            CANAL_X[n*CANAL_M+m][0]=Rx[m][n];
            CANAL_X[n*CANAL_M+m][1]=Ry[m][n];
            CANAL_X[n*CANAL_M+m][2]=Rz[m][n];

            CANAL_V[n*CANAL_M+m][0]=0.0;
            CANAL_V[n*CANAL_M+m][1]=0.0;
            CANAL_V[n*CANAL_M+m][2]=0.0;

            CANAL_residuo[n*CANAL_M+m]=n+1;

            CANAL_chgrp[n*CANAL_M+m]=n*CANAL_M+m+1;
        }
    }

    CANAL_box[0]=fabs(Rx[CANAL_M-1][CANAL_Nb-1]*1.1);
    CANAL_box[1]=fabs(Ry[CANAL_M-1][CANAL_Nb-1]*1.1);
    CANAL_box[2]=fabs(Rz[CANAL_M-1][CANAL_Nb-1]*1.1);

    CANAL_Angle[0]=0.5*PI;
    CANAL_Angle[1]=0;


    ///Liberar memoria
    for(i=0;i<CANAL_M;i++){
        free(*(Rx+i));
        free(*(Ry+i));
        free(*(Rz+i));
    }

    free(Rx);
    free(Ry);
    free(Rz);


    for(i=0;i<(CANAL_M*CANAL_Nb);i++)
        free(*(MDistancia+i));

    free(MDistancia);

    for(i=0;i<CANAL_M*CANAL_Nb;i++)
        free(*(MVecinos+i));
    free(MVecinos);

    for(i=0;i<CANAL_M*CANAL_Nb;i++)
        free(*(MEnlaces+i));
    free(MEnlaces);


    for(i=0;i<CANAL_M*CANAL_Nb;i++)
        for(j=0;j<CANAL_M*CANAL_Nb;j++)
            free(*(*(MAngulos+j)+i));

    for(i=0;i<CANAL_M*CANAL_Nb;i++)
        free(*(MAngulos+i));

    free(MAngulos);

}

void ConstruyeDisco(float **Rx,float **Ry,float **Rz, float CANAL_bond_reposo,float CANAL_radio,
```

```
     float phi, int M, int i){
    int m;
    float phi_inicial=0;

    if(i%2==0)
        phi_inicial=0;
    else
        phi_inicial=0.5*phi;


    for(m=0;m<M;m++){
        Rx[m][i]=CANAL_bond_reposo*i;
        Ry[m][i]=CANAL_radio*cos(m*phi+phi_inicial);
        Rz[m][i]=CANAL_radio*sin(m*phi+phi_inicial);

    }

}


void MatrizDistancia(float **MD,float **Rx,float **Ry, float **Rz,int M, int Nb){
    int i,j,m1,n1,m2,n2;

    for(i=0;i<(M*Nb);i++){
        for(j=0;j<(M*Nb);j++){
            m1=i%M;
            n1=i/M;
            m2=j%M;
            n2=j/M; //Estas cuatro lineas convierten el indice en los valores m,n

            MD[i][j]=Distancia(Rx,Ry,Rz,m1,n1,m2,n2);
        }
    }
}


void MatrizVecinos(int **MV,int M, int Nb,float **MatrizDistancia,float CANAL_bond_reposo){

    int i,j,m1,n1,m2,n2;
    float cota_dist=sqrt(2.0)*CANAL_bond_reposo;

    for(i=0;i<M*Nb;i++){
        for(j=0;j<M*Nb;j++)
            MV[i][j]=0;
    }

    for(i=0;i<M*Nb;i++){
        m1=i%M;
        n1=i/M;
        for(j=0;j<M*Nb;j++){
            m2=j%M;
            n2=j/M;
            if(i!=j && MatrizDistancia[i][j]<cota_dist)
                MV[i][j]=1;


            if(i!=j && n1==n2 && m1==0 && m2==M-2)
                MV[i][j]=2;

            if(i!=j && n1==n2 && m1==1 && m2==M-1)
                MV[i][j]=2;

            if(i!=j && n1==n2 && m1==m2+2)
                MV[i][j]=2;

            if(i!=j && m1==m2-1 && n1==n2-2)
                MV[i][j]=2;

            if(i!=j && m1==m2+1 && n1==n2-2)
                MV[i][j]=2;

            if(i!=j && m1==M-1 && m2==0 && n1==n2-2)
                MV[i][j]=2;

            if(i!=j && m1==0 && m2==M-1 && n1==n2-2)
                MV[i][j]=2;


        }
    }

}

void GeneraBond(int **MatrizEnlaces,int **MVecinos, int M, int Nb){
    int i,j;

    for(i=0;i<M*Nb;i++){
        for(j=0;j<M*Nb;j++)
            MatrizEnlaces[i][j]=0;
    }

    for(i=0;i<M*Nb;i++){
        for(j=0;j<M*Nb;j++){
```

```
            if(MatrizEnlaces[i][j]==0 && MVecinos[i][j]==1){
                MatrizEnlaces[i][j]=1;
                MatrizEnlaces[j][i]=-1; //De este modo marco el enlace inverso para no repetirlo
                    despues!
            }
        }
    }

}

void GeneraAngle(int ***MatrizAngulos,int **MVecinos, int M, int Nb){
    int i,j,k;

    for(i=0;i<M*Nb;i++){
        for(j=0;j<M*Nb;j++){
            for(k=0;k<M*Nb;k++)
                MatrizAngulos[i][j][k]=0;
        }
    }

    for(i=0;i<M*Nb;i++){
        for(j=0;j<M*Nb;j++){
            for(k=0;k<M*Nb;k++){
                if(MatrizAngulos[i][j][k]==0 && MVecinos[i][j]==1 && MVecinos[j][k]==1 && MVecinos[i
                    ][k]==2){
                    MatrizAngulos[i][j][k]=1;
                    MatrizAngulos[i][k][j]=-1;
                    MatrizAngulos[j][i][k]=-1;
                    MatrizAngulos[k][i][j]=-1;
                    MatrizAngulos[j][k][i]=-1;
                    MatrizAngulos[k][j][i]=-1; //De este modo marco el enlace inverso para no
                        repetirlo despues!
                }
            }
        }
    }

}

void ConstruyeTopol_canal(float CANAL_bond_reposo,float CANAL_bond_fuerza, float CANAL_bending_ang,
    float CANAL_bending_fuerza,int M, int Nb,int NExterior,int **MEnlaces, int ***MAngulos, int
    index_simulacion, char* Borde_poro, char* Cuerpo_poro){

    int i,j,k;
    FILE *topol;
    char CanalTopol_name[256];

    sprintf(CanalTopol_name,"SISTEMA_canal_%d.itp",index_simulacion);
    topol=fopen(CanalTopol_name,"w");

    fprintf(topol,";Canal proteico para simulacion de traslocacion de DNA. TFM Guillermo Diez.\n\n")
        ;
    fprintf(topol,"[moleculetype]\n;name\texclusion\nCANAL\t1\n\n");

    fprintf(topol,"[atoms]\n;nr\tattype\tresnr\tresidue\tatom\tcgnr\tcharge\tmass\n");
    for(i=0;i<Nb;i++){
        for(j=0;j<M;j++){
            if(i<NExterior || i+1>Nb-NExterior)
                fprintf(topol,"%d\t%s\t%d\tCHNNL\tQ0\t%d\t%.3f\t;\n",j+M*i+1,Borde_poro,i+1,j+M*i
                    +1,0.0); //OJO! Donde pone "Q0" normalmente pone "P5"!!
            else
                fprintf(topol,"%d\t%s\t%d\tCHNNL\tC1\t%d\t%.3f\t;\n",j+M*i+1,Cuerpo_poro,i+1,j+M*i
                    +1,0.0);
        }
    }


    fprintf(topol,"\n\n");
    fprintf(topol,"[bonds]\n;i\tj\tfunc\tb_0\tk_b\n");
    for(i=0;i<M*Nb;i++){
        for(j=0;j<M*Nb;j++){
            if(MEnlaces[i][j]==1)
                fprintf(topol,"%d\t%d\t%d\t%.3f\t%.1f\n",i+1,j+1,1,CANAL_bond_reposo,
                    CANAL_bond_fuerza);
        }
    }

    fprintf(topol,"\n\n");
    fprintf(topol,"[angles]\n;i\tj\tk\tfunc\ttheta_0\tk_theta\n");
    for(i=0;i<M*Nb;i++){
        for(j=0;j<M*Nb;j++){
            for(k=0;k<M*Nb;k++){
                if(MAngulos[i][j][k]==1)
                    fprintf(topol,"%d\t%d\t%d\t%d\t%.3f\t%.1f\n",i+1,j+1,k+1,1,CANAL_bending_ang,
                        CANAL_bending_fuerza);
            }
        }
    }
    fclose(topol);
}


///Construccion del polimero
```

```
void ConstruyePolimero(int POL_n,float POL_bond_reposo, float POL_bond_fuerza,float POL_bending_ang,
     float POL_bending_fuerza,float **POL_X, float **POL_V, int *POL_residuo, int *POL_chgrp, float
     *POL_box, float *POL_Angle, int index_simulacion ,char* Borde_polimero, char* Cuerpo_polimero){
    int i;
    float VectorDirector[3];

    POL_Angle[0]=0.5*PI;
    POL_Angle[1]=0;


    for(i=0;i<POL_n;i++){
        POL_X[i][0]=i*POL_bond_reposo;
        POL_X[i][1]=0;
        POL_X[i][2]=0;

        POL_V[i][0]=0;
        POL_V[i][1]=0;
        POL_V[i][2]=0;

        POL_residuo[i]=1;

        POL_chgrp[i]=i+1;
    }

    POL_box[0]=1.1*POL_n;
    POL_box[1]=1;
    POL_box[2]=1;

    ConstruyeTopol_polimero(POL_n,POL_bond_reposo,POL_bond_fuerza,POL_bending_ang,POL_bending_fuerza
        ,index_simulacion,Borde_polimero,Cuerpo_polimero);
}

void ConstruyeTopol_polimero(int POL_n,float POL_bond_reposo,float POL_bond_fuerza,float
     POL_bending_ang,float POL_bending_fuerza,int index_simulacion,char* Borde_polimero,char*
     Cuerpo_polimero){
    int i;
    FILE *POL_nuevo_topol;
    char PolTopol_name[256];

    sprintf(PolTopol_name,"SISTEMA_pol_%d.itp",index_simulacion);
    POL_nuevo_topol=fopen(PolTopol_name,"w");

    fprintf(POL_nuevo_topol,";cg PEG9_MOD\n");
    fprintf(POL_nuevo_topol,"[moleculetype]\n");
    fprintf(POL_nuevo_topol,";name\texclusiones\n");
    fprintf(POL_nuevo_topol,"PEG\t1\n");
    fprintf(POL_nuevo_topol,"[atoms]\n");
    fprintf(POL_nuevo_topol,";nr\tattype\tresnr\tresidue\tatom\tcgnr\tcharge\tmass\n");

    for(i=0;i<POL_n;i++){
        if(i==0)
            fprintf(POL_nuevo_topol,"%d\t%s\t%d\t%s\t%s\t%d\t%.3f\t;\n",i+1,Borde_polimero,1,"PEG","
                C",i+1,0.0);
        if(i>0 && i<POL_n-1)
            fprintf(POL_nuevo_topol,"%d\t%s\t%d\t%s\t%s\t%d\t%.3f\t;\n",i+1,Cuerpo_polimero,1,"PEG",
                "C",i+1,0.0);
        if(i==POL_n-1)
            fprintf(POL_nuevo_topol,"%d\t%s\t%d\t%s\t%s\t%d\t%.3f\t;\n",i+1,Borde_polimero,1,"PEG","
                C",i+1,0.0);
    }

    fprintf(POL_nuevo_topol,"[bonds]\n");
    fprintf(POL_nuevo_topol,";ai\taj\tfunc\tb_0\tk_b\n");

    for(i=0;i<POL_n-1;i++)
        fprintf(POL_nuevo_topol,"%d\t%d\t%d\t%.3f\t%.3f\n",i+1,i+2,1,POL_bond_reposo,POL_bond_fuerza
            );

    fprintf(POL_nuevo_topol,"[angles]\n");
    fprintf(POL_nuevo_topol,";ai\taj\tak\tfunc\ttheta_0\tk_theta\n");

    for(i=0;i<POL_n-2;i++)
        fprintf(POL_nuevo_topol,"%d\t%d\t%d\t%d\t%.2f\t%.2f\n",i+1,i+2,i+3,1,POL_bending_ang,
            POL_bending_fuerza);


    fprintf(POL_nuevo_topol,"[dihedrals]\n");
    fprintf(POL_nuevo_topol,";\n");

    for(i=0;i<POL_n-3;i++){
        fprintf(POL_nuevo_topol,"%d\t%d\t%d\t%d\t%d\t%.2f\t%.2f\t%d\n",i+1,i+2,i+3,i
            +4,1,180.00,1.96,1);
        fprintf(POL_nuevo_topol,"%d\t%d\t%d\t%d\t%d\t%.2f\t%.2f\t%d\n",i+1,i+2,i+3,i+4,1,0.0,0.18,2)
            ;
        fprintf(POL_nuevo_topol,"%d\t%d\t%d\t%d\t%d\t%.2f\t%.2f\t%d\n",i+1,i+2,i+3,i+4,1,0.0,0.33,3)
            ;
        fprintf(POL_nuevo_topol,"%d\t%d\t%d\t%d\t%d\t%.2f\t%.2f\t%d\n",i+1,i+2,i+3,i+4,1,0.0,0.12,4)
            ;
    }


    fclose(POL_nuevo_topol);
}
```

```
///Construccion de la membrana
void ConstruyeDSPC(float **DSPC_X, float **DSPC_V, int *DSPC_residuo, int *DSPC_chgrp, float *
     DSPC_box, float *DSPC_Angle, char *DSPC_atomname, float angulo,int AGUA_sel ,int
     index_simulacion, char* campoE){

     int i,DSPC_n=14;

     DSPC_Angle[0]=0.5*PI;
     DSPC_Angle[1]=0;

     angulo=angulo*PI/180;

     DSPC_X[0][0]=0;
     DSPC_X[0][1]=0;
     DSPC_X[0][2]=0;
     DSPC_atomname[0]='N';

     DSPC_X[1][0]=DSPC_X[0][0]+0.47;
     DSPC_X[1][1]=DSPC_X[0][1];
     DSPC_X[1][2]=DSPC_X[0][2];
     DSPC_atomname[1]='P';

     DSPC_X[2][0]=DSPC_X[1][0]+0.47*cos(angulo);
     DSPC_X[2][1]=DSPC_X[1][1]+0.47*sin(angulo);
     DSPC_X[2][2]=DSPC_X[1][2];
     DSPC_atomname[2]='G';

     DSPC_X[3][0]=DSPC_X[1][0]+0.47*cos(angulo);
     DSPC_X[3][1]=DSPC_X[1][1]-0.47*sin(angulo);
     DSPC_X[3][2]=DSPC_X[1][2];
     DSPC_atomname[3]='G';


     DSPC_X[4][0]=DSPC_X[2][0]+0.47*cos(angulo);
     DSPC_X[4][1]=DSPC_X[2][1]+0.47*sin(angulo);
     DSPC_X[4][2]=DSPC_X[2][2];
     DSPC_atomname[4]='C';

     for(i=5;i<9;i++){
          DSPC_X[i][0]=DSPC_X[i-1][0]+0.47;
          DSPC_X[i][1]=DSPC_X[i-1][1];
          DSPC_X[i][2]=DSPC_X[i-1][2];
          DSPC_atomname[i]='C';
     }

     DSPC_X[9][0]=DSPC_X[3][0]+0.47*cos(angulo);
     DSPC_X[9][1]=DSPC_X[3][1]-0.47*sin(angulo);
     DSPC_X[9][2]=DSPC_X[3][2];
     DSPC_atomname[9]='C';

     for(i=10;i<14;i++){
          DSPC_X[i][0]=DSPC_X[i-1][0]+0.47;
          DSPC_X[i][1]=DSPC_X[i-1][1];
          DSPC_X[i][2]=DSPC_X[i-1][2];
          DSPC_atomname[i]='C';
     }



     for(i=0;i<14;i++){
          DSPC_V[i][0]=0;
          DSPC_V[i][1]=0;
          DSPC_V[i][2]=0;

          DSPC_residuo[i]=1;

          DSPC_chgrp[i]=i+1;
     }

     DSPC_box[0]=5.0;
     DSPC_box[1]=1.0;
     DSPC_box[2]=1.0;

     //Construye topologia:
     ConstruyeTopol_DSPC(index_simulacion,AGUA_sel,campoE);
}


void UbicarEnMembrana(float ****MEMBRANA_X, float **DSPC_X, int DSPC_n, int i, int j){
     int k;

     for(k=0;k<DSPC_n;k++){
          MEMBRANA_X[i][j][k][0]=DSPC_X[k][0];
          MEMBRANA_X[i][j][k][1]=DSPC_X[k][1];
          MEMBRANA_X[i][j][k][2]=DSPC_X[k][2];
     }
}

int AsignaOrientacion_Membrana(int i, int j, int MEMBRANA_n){
     int k,l;
```

```
    int **Asignacion;
    int retorno;

    Asignacion=(int**)malloc(sizeof(int*)*MEMBRANA_n);
    for(k=0;k<MEMBRANA_n;k++)
        Asignacion[k]=(int*)malloc(sizeof(int)*MEMBRANA_n);


    Asignacion[0][0]=0;
    for(k=1;k<MEMBRANA_n;k++){
        if(Asignacion[0][k-1]==0)
            Asignacion[0][k]=1;

        if(Asignacion[0][k-1]==1)
            Asignacion[0][k]=0;
    }

    for(k=1;k<MEMBRANA_n;k++){
        if(Asignacion[k-1][0]==0)
            Asignacion[k][0]=1;

        if(Asignacion[k-1][0]==1)
            Asignacion[k][0]=0;
    }


    for(k=1;k<MEMBRANA_n;k++){
        for(l=1;l<MEMBRANA_n;l++){
            if(Asignacion[k-1][l]==0 && Asignacion[k][l-1]==0)
                Asignacion[k][l]=1;

            if(Asignacion[k-1][l]==1 && Asignacion[k][l-1]==1)
                Asignacion[k][l]=0;
        }
    }



    retorno=Asignacion[i][j];

    for(k=0;k<MEMBRANA_n;k++)
        free(Asignacion[k]);
    free(Asignacion);

    return retorno;

} ///Esta funcion devuelve 1 si el acido graso debe estar orientado en sentido +, y 0 si el sentido
    debe ser - (utilizar en el bucle de construccion de la membrana, "rellenar membrana")

int AhuecarMembrana(float ****MEMBRANA_X,int MEMBRANA_n,float radio){
    int i,j,k,l;
    float distancia;
    int eliminados=0;
    float Centro[2]={0,0};

    for(i=0;i<MEMBRANA_n;i++){
        for(j=0;j<MEMBRANA_n;j++){
            Centro[0]+=MEMBRANA_X[i][j][0][0];
            Centro[1]+=MEMBRANA_X[i][j][0][1];
        }
    }

    Centro[0]=Centro[0]/(MEMBRANA_n*MEMBRANA_n);
    Centro[1]=Centro[1]/(MEMBRANA_n*MEMBRANA_n);

    for(i=0;i<MEMBRANA_n;i++){
        for(j=0;j<MEMBRANA_n;j++){
            distancia=sqrt((Centro[0]-MEMBRANA_X[i][j][0][0])*(Centro[0]-MEMBRANA_X[i][j][0][0])+(
                Centro[1]-MEMBRANA_X[i][j][0][1])*(Centro[1]-MEMBRANA_X[i][j][0][1]));
            if(distancia<radio){
                MEMBRANA_X[i][j][0]=NULL;
                eliminados++;
            }
        }
    }
    return eliminados;
}

void ConstruyeTopol_DSPC(int index_simulacion, int AGUA_sel,char* campoE){
    int i;
    FILE *dspc_topol;
    char DSPCTopol_name[256];

    sprintf(DSPCTopol_name,"SISTEMA_dspc_%d.itp",index_simulacion);
    dspc_topol=fopen(DSPCTopol_name,"w");

    fprintf(dspc_topol,";Molecula 'DSPC'[ S.J. Marrink, A.H. de Vries, A.E. Mark] para simulacion de
        traslocacion de DNA. TFM Guillermo Diez.\n\n");

    fprintf(dspc_topol,"[moleculetype]\n;molname\tnrexcl\nDSPC\t%d\n\n",1);
    fprintf(dspc_topol,"[atoms]\n;id\ttype\tresnr\tresidu\tatom\tcgnr\tcharge\n");

    if(strcmp(campoE,"si")==0)
```

```
    {
        fprintf(dspc_topol,"1\tQ0\t1\tDSPC\tNC3\t1\t0.0\n");
        fprintf(dspc_topol,"2\tQa\t1\tDSPC\tPO4\t2\t0.0\n");
    }
    else
    {
        fprintf(dspc_topol,"1\tQ0\t1\tDSPC\tNC3\t1\t1.0\n");
        fprintf(dspc_topol,"2\tQa\t1\tDSPC\tPO4\t2\t-1.0\n");
    }

    fprintf(dspc_topol,"3\tNa\t1\tDSPC\tGL1\t3\t0.0\n");
    fprintf(dspc_topol,"4\tNa\t1\tDSPC\tGL2\t4\t0.0\n");
    fprintf(dspc_topol,"5\tC1\t1\tDSPC\tC1A\t5\t0.0\n");
    fprintf(dspc_topol,"6\tC1\t1\tDSPC\tC2A\t6\t0.0\n");
    fprintf(dspc_topol,"7\tC1\t1\tDSPC\tC3A\t7\t0.0\n");
    fprintf(dspc_topol,"8\tC1\t1\tDSPC\tC4A\t8\t0.0\n");
    fprintf(dspc_topol,"9\tC1\t1\tDSPC\tC5A\t9\t0.0\n");
    fprintf(dspc_topol,"10\tC1\t1\tDSPC\tC1B\t10\t0.0\n");
    fprintf(dspc_topol,"11\tC1\t1\tDSPC\tC2B\t11\t0.0\n");
    fprintf(dspc_topol,"12\tC1\t1\tDSPC\tC3B\t12\t0.0\n");
    fprintf(dspc_topol,"13\tC1\t1\tDSPC\tC4B\t13\t0.0\n");
    fprintf(dspc_topol,"14\tC1\t1\tDSPC\tC5B\t14\t0.0\n\n");

    if(AGUA_sel==1)
    {
        fprintf(dspc_topol,"[bonds]\n;i\tj\tfunct\tlength\tforce.c.\n");
        fprintf(dspc_topol,"%d\t%d\t%d\t%.3f\t%.3f\n",1,2,1,0.47,1250.0);
        fprintf(dspc_topol,"%d\t%d\t%d\t%.3f\t%.3f\n",2,3,1,0.47,1250.0);
        fprintf(dspc_topol,"%d\t%d\t%d\t%.3f\t%.3f\n",3,4,1,0.47,1250.0);
        fprintf(dspc_topol,"%d\t%d\t%d\t%.3f\t%.3f\n",3,5,1,0.47,1250.0);
        fprintf(dspc_topol,"%d\t%d\t%d\t%.3f\t%.3f\n",5,6,1,0.47,1250.0);
        fprintf(dspc_topol,"%d\t%d\t%d\t%.3f\t%.3f\n",6,7,1,0.47,1250.0);
        fprintf(dspc_topol,"%d\t%d\t%d\t%.3f\t%.3f\n",7,8,1,0.47,1250.0);
        fprintf(dspc_topol,"%d\t%d\t%d\t%.3f\t%.3f\n",8,9,1,0.47,1250.0);
        fprintf(dspc_topol,"%d\t%d\t%d\t%.3f\t%.3f\n",4,10,1,0.47,1250.0);
        fprintf(dspc_topol,"%d\t%d\t%d\t%.3f\t%.3f\n",10,11,1,0.47,1250.0);
        fprintf(dspc_topol,"%d\t%d\t%d\t%.3f\t%.3f\n",11,12,1,0.47,1250.0);
        fprintf(dspc_topol,"%d\t%d\t%d\t%.3f\t%.3f\n",12,13,1,0.47,1250.0);
        fprintf(dspc_topol,"%d\t%d\t%d\t%.3f\t%.3f\n\n",13,14,1,0.47,1250.0);

        fprintf(dspc_topol,"[angles]\n;i\tj\tk\tfunct\tangle\tforce.c.\n");

        fprintf(dspc_topol,"%d\t%d\t%d\t%d\t%.3f\t%.3f\n",2,3,4,2,120.0,25.0);
        fprintf(dspc_topol,"%d\t%d\t%d\t%d\t%.3f\t%.3f\n",2,3,5,2,120.0,25.0);
        fprintf(dspc_topol,"%d\t%d\t%d\t%d\t%.3f\t%.3f\n",3,5,6,2,120.0,25.0);
        fprintf(dspc_topol,"%d\t%d\t%d\t%d\t%.3f\t%.3f\n",5,6,7,2,120.0,25.0);
        fprintf(dspc_topol,"%d\t%d\t%d\t%d\t%.3f\t%.3f\n",6,7,8,2,120.0,25.0);
        fprintf(dspc_topol,"%d\t%d\t%d\t%d\t%.3f\t%.3f\n",7,8,9,2,120.0,25.0);
        fprintf(dspc_topol,"%d\t%d\t%d\t%d\t%.3f\t%.3f\n",4,10,11,2,120.0,25.0);
        fprintf(dspc_topol,"%d\t%d\t%d\t%d\t%.3f\t%.3f\n",10,11,12,2,120.0,25.0);
        fprintf(dspc_topol,"%d\t%d\t%d\t%d\t%.3f\t%.3f\n",11,12,13,2,120.0,25.0);
        fprintf(dspc_topol,"%d\t%d\t%d\t%d\t%.3f\t%.3f\n",12,13,14,2,120.0,25.0);
    }

    if(AGUA_sel==0)
    {
        fprintf(dspc_topol,"[bonds]\n;i\tj\tfunct\tlength\tforce.c.\n");
        fprintf(dspc_topol,"%d\t%d\t%d\t%.3f\t%.3f\n",1,2,1,0.45,1250.0);
        fprintf(dspc_topol,"%d\t%d\t%d\t%.3f\t%.3f\n",2,3,1,0.45,1250.0);
        fprintf(dspc_topol,"%d\t%d\t%d\t%.3f\t%.3f\n",3,4,1,0.37,1250.0);
        fprintf(dspc_topol,"%d\t%d\t%d\t%.3f\t%.3f\n",3,5,1,0.48,1250.0);
        fprintf(dspc_topol,"%d\t%d\t%d\t%.3f\t%.3f\n",5,6,1,0.48,1250.0);
        fprintf(dspc_topol,"%d\t%d\t%d\t%.3f\t%.3f\n",6,7,1,0.48,1250.0);
        fprintf(dspc_topol,"%d\t%d\t%d\t%.3f\t%.3f\n",7,8,1,0.48,1250.0);
        fprintf(dspc_topol,"%d\t%d\t%d\t%.3f\t%.3f\n",8,9,1,0.48,1250.0);
        fprintf(dspc_topol,"%d\t%d\t%d\t%.3f\t%.3f\n",4,10,1,0.48,1250.0);
        fprintf(dspc_topol,"%d\t%d\t%d\t%.3f\t%.3f\n",10,11,1,0.48,1250.0);
        fprintf(dspc_topol,"%d\t%d\t%d\t%.3f\t%.3f\n",11,12,1,0.48,1250.0);
        fprintf(dspc_topol,"%d\t%d\t%d\t%.3f\t%.3f\n",12,13,1,0.48,1250.0);
        fprintf(dspc_topol,"%d\t%d\t%d\t%.3f\t%.3f\n\n",13,14,1,0.48,1250.0);

        fprintf(dspc_topol,"[angles]\n;i\tj\tk\tfunct\tangle\tforce.c.\n");

        fprintf(dspc_topol,"%d\t%d\t%d\t%d\t%.3f\t%.3f\n",2,3,4,2,120.0,25.0);
        fprintf(dspc_topol,"%d\t%d\t%d\t%d\t%.3f\t%.3f\n",2,3,5,2,180.0,25.0);
        fprintf(dspc_topol,"%d\t%d\t%d\t%d\t%.3f\t%.3f\n",3,5,6,2,180.0,35.0);
        fprintf(dspc_topol,"%d\t%d\t%d\t%d\t%.3f\t%.3f\n",5,6,7,2,180.0,35.0);
        fprintf(dspc_topol,"%d\t%d\t%d\t%d\t%.3f\t%.3f\n",6,7,8,2,180.0,35.0);
        fprintf(dspc_topol,"%d\t%d\t%d\t%d\t%.3f\t%.3f\n",7,8,9,2,180.0,35.0);
        fprintf(dspc_topol,"%d\t%d\t%d\t%d\t%.3f\t%.3f\n",4,10,11,2,180.0,35.0);
        fprintf(dspc_topol,"%d\t%d\t%d\t%d\t%.3f\t%.3f\n",10,11,12,2,180.0,35.0);
        fprintf(dspc_topol,"%d\t%d\t%d\t%d\t%.3f\t%.3f\n",11,12,13,2,180.0,35.0);
        fprintf(dspc_topol,"%d\t%d\t%d\t%d\t%.3f\t%.3f\n",12,13,14,2,180.0,35.0);

    }


    fclose(dspc_topol);

}


///Desplazamiento de estructuras
```

```
void MoverEstructura(float **estructura,int N,float theta, float phi, float Rx, float Ry, float Rz,
     long idum, float *Angle){ ///Mueve una estructura a la orientacion (theta,phi), posicion R
     int i;
     float CentroMasa[3]={0,0,0};
     float delta_theta, delta_phi;
     float Eje_rot_phi[3]={0,0,1};
     float Eje_rot_theta[3];
     float NuevaDireccion[3]={sin(theta)*cos(phi),sin(theta)*sin(phi),cos(theta)};
     float VectorDirector[3];
     float Mod=0;

     Calcula_CM(estructura,N,CentroMasa);

     VectorDirector[0]=sin(Angle[0])*cos(Angle[1]);
     VectorDirector[1]=sin(Angle[0])*sin(Angle[1]);
     VectorDirector[2]=cos(Angle[0]);


     delta_theta=theta-180*Angle[0]/PI;
     delta_phi=phi-180*Angle[1]/PI;


     Rotacion(Eje_rot_phi,VectorDirector,delta_phi);
     for(i=0;i<N;i++)
          Rotacion(Eje_rot_phi,*(estructura+i),delta_phi);

     ProductoVectorial(VectorDirector,NuevaDireccion,Eje_rot_theta);
     Mod=sqrt(ProductoEscalar(Eje_rot_theta,Eje_rot_theta));
     if(Mod<0.00001 && Mod >-0.00001){
          Mod=1.0;
          Eje_rot_theta[0]=NuevaDireccion[0];
          Eje_rot_theta[1]=NuevaDireccion[1];
          Eje_rot_theta[2]=NuevaDireccion[2];
     }

     Eje_rot_theta[0]=Eje_rot_theta[0]/Mod;
     Eje_rot_theta[1]=Eje_rot_theta[1]/Mod;
     Eje_rot_theta[2]=Eje_rot_theta[2]/Mod;


     Rotacion(Eje_rot_theta,VectorDirector,delta_theta);
     for(i=0;i<N;i++)
          Rotacion(Eje_rot_theta,*(estructura+i),delta_theta);

     for(i=0;i<N;i++){
          estructura[i][0]=estructura[i][0]+Rx;
          estructura[i][1]=estructura[i][1]+Ry;
          estructura[i][2]=estructura[i][2]+Rz;
     }


     Angle[0]=PI/180*theta;
     Angle[1]=PI/180*phi;

} ///Coloca el CM de "estructura" en Rx,Ry,Rz; y la orienta en thtea, phi.

void Calcula_CM(float **estructura, int N, float *CentroMasa){
     int i;

     CentroMasa[0]=0;
     CentroMasa[1]=0;
     CentroMasa[2]=0;


     for(i=0;i<N;i++){
          CentroMasa[0]+=estructura[i][0];
          CentroMasa[1]+=estructura[i][1];
          CentroMasa[2]+=estructura[i][2];
     }

     CentroMasa[0]=CentroMasa[0]/((float)N);
     CentroMasa[1]=CentroMasa[1]/((float)N);
     CentroMasa[2]=CentroMasa[2]/((float)N); ///Aqui esta calculado el centro de masas.

     for(i=0;i<N;i++){
          estructura[i][0]=estructura[i][0]-CentroMasa[0];
          estructura[i][1]=estructura[i][1]-CentroMasa[1];
          estructura[i][2]=estructura[i][2]-CentroMasa[2];
     }
}


///Construccion de indices y topologia
void ConstruyeIndices(int SISTEMA_n, int CANAL_n,int CANAL_M,int CANAL_Nb,float CANAL_bond_reposo,
     int POL_n,int DSPC_n ,int NumeroDSPC, int index_simulacion,int CANAL_sel,int POL_sel, float
     POL_posicion,float POL_bond_reposo, int MEMBRANA_sel){
     int i;
     int *N_pol_dentro;
     FILE *punte;
     char Indices_name[256];

     N_pol_dentro=(int*)malloc(sizeof(int)*POL_n);
     for(i=0;i<POL_n;i++)
```

```
    {
        if(fabs((POL_n-i)*POL_bond_reposo-0.5*(POL_n+1)*POL_bond_reposo+0.5*POL_posicion*
            POL_bond_reposo*POL_n)<CANAL_bond_reposo*0.5*CANAL_Nb)
            N_pol_dentro[i]=1;
        else
            N_pol_dentro[i]=-1;
    }

    sprintf(Indices_name,"SISTEMA_index_%d.ndx",index_simulacion);
    punte=fopen(Indices_name,"w");

    if(CANAL_sel!=0)
    {
        fprintf(punte, "[CHNNL]\n");
        for(i=0;i<CANAL_n;i++)
            fprintf(punte,"%d\n",i+1);
    }

    if(POL_sel!=0)
    {
        fprintf(punte, "\n[PEG]\n");
        for(i=0;i<POL_n;i++)
            fprintf(punte,"%d\n",CANAL_sel*CANAL_n+i+1);
    }

    if(MEMBRANA_sel!=0)
    {
        fprintf(punte, "\n[DSPC]\n");
        for(i=0;i<NumeroDSPC*DSPC_n;i++)
            fprintf(punte,"%d\n",CANAL_sel*CANAL_n+POL_sel*POL_n+i+1);
    }

    if(POL_sel!=0)
    {
        fprintf(punte, "\n[PULLING]\n");
        fprintf(punte,"%d\n",CANAL_sel*CANAL_n+1);
    }


    fprintf(punte, "\n[SECO]\n");
    for(i=0;i<SISTEMA_n;i++)
        fprintf(punte,"%d\n",i+1);

    if(CANAL_sel!=0 || POL_sel!=0)
    {
        fprintf(punte, "\n[ANALISIS]\n");
        for(i=0;i<CANAL_M;i++)
            fprintf(punte,"%d\n",i+1);
        fprintf(punte,"%d\n",CANAL_sel*CANAL_n+POL_sel*POL_n);
    }

    if(CANAL_sel!=0 || POL_sel!=0)
    {
        fprintf(punte, "\n[CONTRAPESO]\n");
        for(i=0;i<CANAL_n;i++)
            fprintf(punte,"%d\n",i+1);
        for(i=0;i<NumeroDSPC*DSPC_n;i++)
            fprintf(punte,"%d\n",CANAL_sel*CANAL_n+POL_sel*POL_n+i+1);
    }

    if(POL_sel!=0)
    {
        fprintf(punte, "\n[DENTRO]\n");
        for(i=0;i<POL_n;i++)
        {
            if(N_pol_dentro[i]==1)
                fprintf(punte,"%d\n",CANAL_sel*CANAL_n+i+1);
        }

    }

    if(POL_sel!=0)
    {
        fprintf(punte, "\n[FUERA]\n");
        for(i=0;i<POL_n;i++)
        {
            if(N_pol_dentro[i]==-1)
                fprintf(punte,"%d\n",CANAL_sel*CANAL_n+i+1);
        }
    }

    fclose(punte);
    free(N_pol_dentro);
}

void EscribeFicheroTopol(int NumeroDSPC, int NumeroW, int index_simulacion, int CANAL_sel, int
    POL_sel, int MEMBRANA_sel){
    FILE *topol;
    char Topol_name[256];

    sprintf(Topol_name,"topol_%d.top",index_simulacion);

    topol=fopen(Topol_name,"w");
```

```
    if(NumeroW>0)
        fprintf(topol,"#include %cmartini_v2.2.itp%c\n",34,34);
    if(NumeroW==0)
        fprintf(topol,"#include %cdry_martini_v2.1.itp%c\n",34,34);
    if(CANAL_sel!=0)
        fprintf(topol,"#include %cSISTEMA_canal_%d.itp%c\n",34,index_simulacion,34);
    if(POL_sel!=0)
        fprintf(topol,"#include %cSISTEMA_pol_%d.itp%c\n",34,index_simulacion,34);
    if(MEMBRANA_sel!=0)
        fprintf(topol,"#include %cSISTEMA_dspc_%d.itp%c\n\n",34,index_simulacion,34);

    fprintf(topol,"[system]\nCanalProteico\n\n[molecules]\n");
    if(CANAL_sel!=0)
        fprintf(topol,"CANAL\t1\n");
    if(POL_sel!=0)
        fprintf(topol,"PEG\t1\n");
    if(MEMBRANA_sel!=0)
        fprintf(topol,"DSPC\t%d",NumeroDSPC);

    if(NumeroW>0)
        fprintf(topol,"\nW\t%d",NumeroW);


    fclose(topol);

}


///CONFIGURACION DEL EXPERIMENTO//////////////////////////////
void Genconf()
{
    int i,j;
    FILE* genconf;
    FILE* Experimento;
    FILE* pipe;

    int N_simulaciones;
    int N_estadistica;
    float term_tol_derivada;
    float term_tolerancia;
    int N_term;
    int N_Lterm;
    float N_sim_cte;

    char AGUA[256];
    char CANAL[256];
    char POLIMERO[256];
    char MEMBRANA[256];
    char pulling[256];
    char PintaTrayectoria[256];
    char Borde_poro[256];
    char Cuerpo_poro[256];
    char Cuerpo_polimero[256];
    char Borde_polimero[256];
    char Acoplo_presion[256];
    char pulling_modo[256];
    char conf_externa[256];
    char campo_E[256];
    char Auto_genconf[256];
    char basura[256];

    float CANAL_bond_fuerza_ini;
    float CANAL_bond_fuerza_fin;
    float d_CANAL_bond_fuerza;

    float CANAL_bond_reposo;

    float CANAL_bending_fuerza_ini;
    float CANAL_bending_fuerza_fin;
    float d_CANAL_bending_fuerza;

    float CANAL_bending_ang;

    float POL_longitud_ini;
    float POL_longitud_fin;
    float d_POL_longitud;

    float pull_rate_ini;
    float pull_rate_fin;
    float d_pull_rate;

    float pull_fuerza_ini;
    float pull_fuerza_fin;
    float d_pull_fuerza;

    float Ezfrecuencia_ini;
    float Ezfrecuencia_fin;
    float d_Ezfrecuencia;

    float Ezintensidad;
    float Exintensidad;
    float Eyintensidad;
```

```
float CANAL_radio;
float POL_posicion;
float POL_bond_fuerza;
float POL_bond_reposo;
float POL_bending_fuerza;
float POL_bending_ang;
float MEMBRANA_area;
float pull_k,temperatura,tau_t;
float CANAL_long_int;
int CANAL_n_ext;



pipe=popen("bash","w");
fprintf(pipe,"mkdir -p ./Input\n");
fflush(pipe);
pclose(pipe);



Experimento=fopen("ExperimentoTraslocacion.txt","r");


fscanf(Experimento,"%s %d",basura,basura,&N_simulaciones);
fscanf(Experimento,"%s %d",basura,basura,&N_estadistica);

fscanf(Experimento,"%s %s",basura,basura,PintaTrayectoria);
fscanf(Experimento,"%s %s",basura,basura,conf_externa);


fscanf(Experimento,"%s %s",basura,basura,CANAL);
fscanf(Experimento,"%s %s",basura,basura,POLIMERO);
fscanf(Experimento,"%s %s",basura,basura,MEMBRANA);
fscanf(Experimento,"%s %s",basura,basura,AGUA);

fscanf(Experimento,"%s %f",basura,basura,&CANAL_radio);
fscanf(Experimento,"%s %f",basura,basura,&CANAL_long_int);
fscanf(Experimento,"%s %d",basura,basura,&CANAL_n_ext);

fscanf(Experimento,"%s %f",basura,basura,&POL_posicion);
fscanf(Experimento,"%s %f",basura,basura,&POL_bond_fuerza);
fscanf(Experimento,"%s %f",basura,basura,&POL_bond_reposo);
fscanf(Experimento,"%s %f",basura,basura,&POL_bending_fuerza);
fscanf(Experimento,"%s %f",basura,basura,&POL_bending_ang);
fscanf(Experimento,"%s %f",basura,basura,&MEMBRANA_area);

fscanf(Experimento,"%s %s",basura,basura,Borde_poro);
fscanf(Experimento,"%s %s",basura,basura,Cuerpo_poro);
fscanf(Experimento,"%s %s",basura,basura,Borde_polimero);
fscanf(Experimento,"%s %s",basura,basura,Cuerpo_polimero);

fscanf(Experimento,"%s %f",basura,basura,&temperatura);
fscanf(Experimento,"%s %f",basura,basura,&tau_t);
fscanf(Experimento,"%s %s",basura,basura,Acoplo_presion);

fscanf(Experimento,"%s %f",basura,basura,&term_tol_derivada);
fscanf(Experimento,"%s %f",basura,basura,&term_tolerancia);
fscanf(Experimento,"%s %d",basura,basura,&N_term);
fscanf(Experimento,"%s %d",basura,basura,&N_Lterm);
fscanf(Experimento,"%s %f",basura,basura,&N_sim_cte);

fscanf(Experimento,"%s %s",basura,basura,pulling);
fscanf(Experimento,"%s %s",basura,basura,pulling_modo);
fscanf(Experimento,"%s %f",basura,basura,&pull_k);
fscanf(Experimento,"%s %f",basura,basura,&pull_rate_ini);
fscanf(Experimento,"%s %f",basura,basura,&pull_rate_fin);
fscanf(Experimento,"%s %f",basura,basura,&pull_fuerza_ini);
fscanf(Experimento,"%s %f",basura,basura,&pull_fuerza_fin);

fscanf(Experimento,"%s %s",basura,basura,campo_E);
fscanf(Experimento,"%s %f",basura,basura,&Ezintensidad);
fscanf(Experimento,"%s %f",basura,basura,&Exintensidad);
fscanf(Experimento,"%s %f",basura,basura,&Eyintensidad);
fscanf(Experimento,"%s %f",basura,basura,&Ezfrecuencia_ini);
fscanf(Experimento,"%s %f",basura,basura,&Ezfrecuencia_fin);

fscanf(Experimento,"%s %f",basura,basura,&CANAL_bond_fuerza_ini);
fscanf(Experimento,"%s %f",basura,basura,&CANAL_bond_fuerza_fin);

fscanf(Experimento,"%s %f",basura,basura,&CANAL_bond_reposo);

fscanf(Experimento,"%s %f",basura,basura,&CANAL_bending_fuerza_ini);
fscanf(Experimento,"%s %f",basura,basura,&CANAL_bending_fuerza_fin);

fscanf(Experimento,"%s %f",basura,basura,&CANAL_bending_ang);

fscanf(Experimento,"%s %f",basura,basura,&POL_longitud_ini);
fscanf(Experimento,"%s %f",basura,basura,&POL_longitud_fin);

fscanf(Experimento,"%s %s",basura,basura,Auto_genconf);

fclose(Experimento);
```

```
if(strcmp(Auto_genconf,"no")==0)
    return;


pull_rate_ini=pull_rate_ini*POL_bond_reposo;
pull_rate_fin=pull_rate_fin*POL_bond_reposo;

genconf=fopen("./Input/genconf.txt","w");

if(N_simulaciones>1)
{
    d_CANAL_bond_fuerza=exp(log(CANAL_bond_fuerza_fin/CANAL_bond_fuerza_ini)/(N_simulaciones-1))
        ;
    d_CANAL_bending_fuerza=exp(log(CANAL_bending_fuerza_fin/CANAL_bending_fuerza_ini)/(
        N_simulaciones-1));
    d_pull_rate=exp(log(pull_rate_fin/pull_rate_ini)/(N_simulaciones-1));
    d_pull_fuerza=exp(log(pull_fuerza_fin/pull_fuerza_ini)/(N_simulaciones-1));
    d_Ezfrecuencia=exp(log(Ezfrecuencia_fin/Ezfrecuencia_ini)/(N_simulaciones-1));
    d_POL_longitud=exp(log(POL_longitud_fin/POL_longitud_ini)/(N_simulaciones-1));
}
else
{
    d_CANAL_bond_fuerza=1;
    d_CANAL_bending_fuerza=1;
    d_pull_rate=1;
    d_pull_fuerza=1;
    d_Ezfrecuencia=1;
    d_POL_longitud=1;
}


fprintf(genconf,"N_simulaciones\t=\t%d\n",N_simulaciones);

fprintf(genconf,"N_estadistica\t=\t%d\n",N_estadistica);

fprintf(genconf,"term_tol_derivada\t=\t%f\n",term_tol_derivada);
fprintf(genconf,"term_tolerancia\t=\t%f\n",term_tolerancia);
fprintf(genconf,"N_Lterm\t=\t%d\n",N_term);
fprintf(genconf,"N_Lterm\t=\t%d\n",N_Lterm);
fprintf(genconf,"N_sim_cte\t=\t%f\n",N_sim_cte);

fprintf(genconf,"PintarTrayectoria\t=\t%s\n",PintaTrayectoria);
fprintf(genconf,"ConfigExterna\t=\t%s\n",conf_externa);

fprintf(genconf,"AcoploPresion\t=\t%s\n",Acoplo_presion);

fprintf(genconf,"BordePoro\t=\t%s\n",Borde_poro);
fprintf(genconf,"CuerpoPoro\t=\t%s\n",Cuerpo_poro);
fprintf(genconf,"BordePolimero\t=\t%s\n",Borde_polimero);
fprintf(genconf,"CuerpoPolimero\t=\t%s\n",Cuerpo_polimero);

fprintf(genconf,"Pulling\t=\t%s\n",pulling);
fprintf(genconf,"Pulling_modo\t=\t%s\n",pulling_modo);

fprintf(genconf,"Pull_rate\t=");


for(i=0;i<N_simulaciones;i++)
    fprintf(genconf,"\t%f",pull_rate_ini*pow(d_pull_rate,i));

fprintf(genconf,"\n");

fprintf(genconf,"Pull_fuerza\t=");


for(i=0;i<N_simulaciones;i++)
    fprintf(genconf,"\t%f",pull_fuerza_ini*pow(d_pull_fuerza,i));

fprintf(genconf,"\n");

fprintf(genconf,"Pull_k\t=\t%f\n",pull_k);
fprintf(genconf,"Temperatura\t=\t%f\n",temperatura);
fprintf(genconf,"tau_t\t=\t%f\n",tau_t);

fprintf(genconf,"AGUA\t=\t%s\n",AGUA);

fprintf(genconf,"CANAL\t=\t%s\n",CANAL);

fprintf(genconf,"CANAL_radio\t=\t%f\n",CANAL_radio);

fprintf(genconf,"CANAL_longitud_Interior\t=\t%f\n",CANAL_long_int);

fprintf(genconf,"CANAL_longitudExterior\t=\t%f\n",CANAL_bond_reposo*CANAL_n_ext);

fprintf(genconf,"CANAL_bond_fuerza\t=");

for(i=0;i<N_simulaciones;i++)
    fprintf(genconf,"\t%f",CANAL_bond_fuerza_ini*pow(d_CANAL_bond_fuerza,i));

fprintf(genconf,"\n");
```

```
    fprintf(genconf,"CANAL_bond_reposo\t=\t%f\n",CANAL_bond_reposo);

    fprintf(genconf,"CANAL_bending\t=");

    for(i=0;i<N_simulaciones;i++)
        fprintf(genconf,"\t%f",CANAL_bending_fuerza_ini*pow(d_CANAL_bending_fuerza,i));

    fprintf(genconf,"\n");

    fprintf(genconf,"CANAL_bending_ang\t=\t%f\n",CANAL_bending_ang);

    fprintf(genconf,"POLIMERO\t=\t%s\n",POLIMERO);

    fprintf(genconf,"POL_bond_fuerza\t=\t%f\n",POL_bond_fuerza);

    fprintf(genconf,"POL_bond_reposo\t=\t%f\n",POL_bond_reposo);


    fprintf(genconf,"POL_bending\t=\t%f\n",POL_bending_fuerza);

    fprintf(genconf,"POL_bending_ang\t=\t%f\n",POL_bending_ang);


    fprintf(genconf,"POL_longitud\t=");
    for(i=0;i<N_simulaciones;i++)
        fprintf(genconf,"\t%f",POL_longitud_ini*pow(d_POL_longitud,i));
    fprintf(genconf,"\n");


    fprintf(genconf,"POL_posicion\t=\t%f\n",POL_posicion);


    fprintf(genconf,"MEMBRANA\t=\t%s\n",MEMBRANA);

    fprintf(genconf,"MEMBRANA_area\t=\t%f\n",MEMBRANA_area);

    fprintf(genconf,"Campo_E\t=\t%s\n",campo_E);
    fprintf(genconf,"Ezfrecuencia\t=");
    for(i=0;i<N_simulaciones;i++)
        fprintf(genconf,"\t%f",Ezfrecuencia_ini*pow(d_Ezfrecuencia,i));
    fprintf(genconf,"\n");

    fprintf(genconf,"Ezintensidad_z\t=\t%f\n",Ezintensidad);
    fprintf(genconf,"Ezintensidad_z\t=\t%f\n",Exintensidad);
    fprintf(genconf,"Ezintensidad_z\t=\t%f\n",Eyintensidad);



    fclose(genconf);



}


///ANALISIS//////////////////////////////////////////////////////////////////////////
float CalculaRadioGiro(int index_simulacion, char* Carpeta, char* GROfile)
{
    int i,j;
    int N_particulas=0;
    float** Coordenadas;
    float RadioVector_medio[3]={0,0,0};
    float Delta_RadioVector[3];
    float radio_giro=0;
    char coord_data_name[256];
    char index_name[256];
    char seeker[256];
    char basura[256];
    char flag0='a', flag1='b';
    int indice_index;
    int indice_particula;
    FILE *coord_data;
    FILE * index;

    sprintf(index_name,"SISTEMA_index_%d.ndx",index_simulacion);
    index=fopen(index_name,"r");

    for(i=0;i<1;i++)
    {
        fscanf(index,"%s",seeker);

        if(strcmp(seeker,"[FUERA]")==0)
        {
            fseek(index,1,SEEK_CUR);
            for(j=0;j<1;j++)
            {
                if(N_particulas==0)
                    fscanf(index,"%d",&indice_index);
                else
                    fscanf(index,"%s",seeker);

                N_particulas++;
```

```
                flag0=fgetc(index);
                flag1=fgetc(index);
                fseek(index,-2,SEEK_CUR);
                if(flag1=='\n' || (flag1==EOF))
                    break;
                j=-1;
            }
            break;
        }
        i=-1;
    } //Aqui ya he contado cuantas particulas contribuyen para calcular el radio de giro

    Coordenadas=(float**)malloc(sizeof(float*)*N_particulas);
    for(i=0;i<N_particulas;i++)
        Coordenadas[i]=(float*)malloc(sizeof(float)*3);

    fclose(index);

    sprintf(coord_data_name,"%s/%s_%d.gro",Carpeta,GROfile,index_simulacion);
    coord_data=fopen(coord_data_name,"r");
    fscanf(coord_data,"%s %s",basura,basura);

    for(i=0;i<1;i++)
    {
        fscanf(coord_data,"%s %s %d %s %s %s %s %s",basura,basura,&indice_particula,basura,basura,basura,
            basura,basura,basura);
        if(indice_particula==indice_index-1)
        {
            for(j=0;j<N_particulas;j++)
                fscanf(coord_data,"%s %s %s %f %f %f %s %s %s",basura,basura,basura,&Coordenadas[j][0],&
                    Coordenadas[j][1],&Coordenadas[j][2],basura,basura,basura);
            break;
        }

        i=-1;
    } //Aqui ya estan almacenadas las coordenas para calcular el radio de giro


    for(i=0;i<N_particulas;i++)
    {
        RadioVector_medio[0]+=Coordenadas[i][0];
        RadioVector_medio[1]+=Coordenadas[i][1];
        RadioVector_medio[2]+=Coordenadas[i][2];
    }
    RadioVector_medio[0]=RadioVector_medio[0]/N_particulas;
    RadioVector_medio[1]=RadioVector_medio[1]/N_particulas;
    RadioVector_medio[2]=RadioVector_medio[2]/N_particulas;

    for(i=0;i<N_particulas;i++)
    {
        Delta_RadioVector[0]=Coordenadas[i][0]-RadioVector_medio[0];
        Delta_RadioVector[1]=Coordenadas[i][1]-RadioVector_medio[1];
        Delta_RadioVector[2]=Coordenadas[i][2]-RadioVector_medio[2];

        radio_giro+=ProductoEscalar(Delta_RadioVector,Delta_RadioVector);
    }
    radio_giro=sqrt(radio_giro/N_particulas);


    for(i=0;i<N_particulas;i++)
        free(Coordenadas[i]);
    free(Coordenadas);
    fclose(coord_data);

    return radio_giro;
}

float TiempoTraslocacion(struct CANAL_struct* CANAL_punt,struct POL_struct* POL_punt, struct
    MEMBRANA_struct* MEMBRANA_punt,struct VARIABLE_struct* VARIABLE_punt,int index_simulacion, int
    index_estad)
{
    int i,j;
    int frames;
    char tiempo_extra_name[256];
    char fichero_analisis[256];
    char basura[256];
    char comando[8192];
    FILE *pipe;
    FILE* read_analisis,*tiempo_extra_file;
    float CM[3];
    int CANAL_M;
    float X_marcador,Y_marcador,Z_marcador;
    float X,Y,Z;
    float tiempo;
    float tiempo_previo;
    float tiempo_sim=0;
    float tiempo_extra;
    float CANAL_radio;

    tiempo_extra=VARIABLE_punt->tiempo_extra;
    CANAL_radio=CANAL_punt->radio;
```

```
    frames=0;
    sprintf(fichero_analisis,"traj_analisis_%d_%d.gro",index_simulacion,index_estad);
    read_analisis=fopen(fichero_analisis,"r");

    for(i=0;i<1;i++)
    {
        if(fgetc(read_analisis)=='=')
            frames++;
        if(feof(read_analisis)!=0)
            break;
        i=-1;
    }

    fseek(read_analisis,0,SEEK_SET);
    for(j=0;j<frames;j++)
    {
        CM[0]=0;
        CM[1]=0;
        CM[2]=0;
        tiempo_previo=tiempo;

        //Leo el tiempo:
        for(i=0;i<1;i++)
        {
            if(fgetc(read_analisis)=='=')
            {
                fscanf(read_analisis,"%f",&tiempo);
                break;
            }
            i=-1;
        }

        //Posicion del CM del disco:
        fscanf(read_analisis,"%d",&CANAL_M);
        CANAL_M=CANAL_M-1;
        for(i=0;i<CANAL_M;i++)
        {
            fscanf(read_analisis,"%s %s %s",basura,basura,basura);
            fscanf(read_analisis,"%f %f %f",&X,&Y,&Z);
            if(j==0 || j==frames-1)
                fscanf(read_analisis,"%s %s %s",basura,basura,basura);

            CM[0]+=X;
            CM[1]+=Y;
            CM[2]+=Z;
        }

        CM[0]=CM[0]/CANAL_M;
        CM[1]=CM[1]/CANAL_M;
        CM[2]=CM[2]/CANAL_M;

        //Posicion del marcador:
        fscanf(read_analisis,"%s %s %s",basura,basura,basura);
        fscanf(read_analisis,"%f %f %f",&X_marcador,&Y_marcador,&Z_marcador);

        if(tiempo>0 && Z_marcador>CM[2] && X_marcador<CM[0]+CANAL_radio && X_marcador>CM[0]-
            CANAL_radio && Y_marcador<CM[1]+CANAL_radio && Y_marcador>CM[1]-CANAL_radio) //Ha
            traslocado
        {
            tiempo_sim=0.5*(tiempo+tiempo_previo);
            break;
        }
    }
    fclose(read_analisis);

    if(fork()==0)
    {
        sprintf(comando,"rm traj_analisis_%d_%d.gro\n",index_simulacion,index_estad);

        pipe=popen(comando,"w");
        pclose(pipe);

        LiberarMemoriaReservada(CANAL_punt,POL_punt,MEMBRANA_punt,VARIABLE_punt); exit(0);
    }
    wait(NULL);

    if(Igual(tiempo_sim,0)==1)
        return 0;
    else
        return tiempo_sim+tiempo_extra;
}


void AnalisisDeResultados(struct CANAL_struct* CANAL_punt,struct POL_struct* POL_punt, struct
    MEMBRANA_struct* MEMBRANA_punt,struct VARIABLE_struct* VARIABLE_punt)
{
    int i,j;
    int index_simulacion,index_estad;
    int N_lineas=0;
    int **puntero_pullx_basura;
    int **puntero_pullf_basura;
    float* estadistica_vel,*estadistica_tiempo,*estadistica_pos;
    float tiempo_trans;
```

```
float vel_medio,tiempo_medio,fuerza_medio;
float error_vel,error_tiempo,error_fuerza;
float POL_long_trans;
float x,y,xf,xp,yf,yp,fuerza_inst=0;
float *array_x;
float *array_W;
float **array_fuerza;
float **array_fuerza_media;
float **array_fuerza_error;
float **array_pos;
float **array_pos_media;
float **array_pos_error;
int fuerza_contador=0;
char flag;
char comando[8192];
char write_analisis_name[256];
char read_analisis_name[256];
char read_fuerza_name[256];
char read_pos_name[256];
FILE *write_analisis,*read_analisis,*pipe,*read_fuerza,*write_trayectoria,*read_pos;

puntero_pullf_basura=(int**)malloc(sizeof(int*)*VARIABLE_punt->N_simulaciones);
for(index_simulacion=0;index_simulacion<VARIABLE_punt->N_simulaciones;index_simulacion++)
    puntero_pullf_basura[index_simulacion]=(int*)malloc(sizeof(int)*VARIABLE_punt->N_estadistica
        );

puntero_pullx_basura=(int**)malloc(sizeof(int*)*VARIABLE_punt->N_simulaciones);
for(index_simulacion=0;index_simulacion<VARIABLE_punt->N_simulaciones;index_simulacion++)
    puntero_pullx_basura[index_simulacion]=(int*)malloc(sizeof(int)*VARIABLE_punt->N_estadistica
        );


printf("\n");
for(index_simulacion=0;index_simulacion<VARIABLE_punt->N_simulaciones;index_simulacion++)
{
    for(index_estad=0;index_estad<VARIABLE_punt->N_estadistica;index_estad++)
    {
        sprintf(read_fuerza_name,"pullf_%d_%d.xvg",index_simulacion,index_estad);
        read_fuerza=fopen(read_fuerza_name,"r");
        for(i=0;i<1;i++)
        {
            flag=fgetc(read_fuerza);
            if(flag=='#' || flag=='@')
            {
                for(j=0;j<1;j++)
                {
                    flag=fgetc(read_fuerza);
                    if(flag=='\n')
                        break;
                    j=-1;
                }
            }
            else
            {
                puntero_pullf_basura[index_simulacion][index_estad]=ftell(read_fuerza)-1;
                for(j=0;j<1;j++)
                {
                    flag=fgetc(read_fuerza);
                    if(flag=='\n' && index_simulacion==0 && index_estad==0)
                        N_lineas++;
                    if(feof(read_fuerza)!=0)
                        break;
                    j=-1;
                }
                break;
            }
            i=-1;
        }
        fclose(read_fuerza);

        sprintf(read_pos_name,"pullx_%d_%d.xvg",index_simulacion,index_estad);
        read_pos=fopen(read_pos_name,"r");
        for(i=0;i<1;i++)
        {
            flag=fgetc(read_pos);
            if(flag=='#' || flag=='@')
            {
                for(j=0;j<1;j++)
                {
                    flag=fgetc(read_pos);
                    if(flag=='\n')
                        break;
                    j=-1;
                }
            }
            else
            {
                puntero_pullx_basura[index_simulacion][index_estad]=ftell(read_pos)-1;
                break;
            }
            i=-1;
        }
        fclose(read_pos);
```

```
            printf("Analisis trayectoria: %.3f %%\n",25.0*(index_simulacion*VARIABLE_punt->
                N_estadistica+index_estad+1.0)/(1.0*VARIABLE_punt->N_simulaciones*VARIABLE_punt->
                N_estadistica));
        }
    }

    array_x=(float*)malloc(sizeof(float)*N_lineas);

    array_W=(float*)malloc(sizeof(float)*VARIABLE_punt->N_simulaciones);

    array_fuerza=(float**)malloc(sizeof(float*)*N_lineas);
    for(i=0;i<N_lineas;i++)
        array_fuerza[i]=(float*)malloc(sizeof(float)*VARIABLE_punt->N_estadistica);

    array_fuerza_media=(float**)malloc(sizeof(float*)*VARIABLE_punt->N_simulaciones);
    for(index_simulacion=0;index_simulacion<VARIABLE_punt->N_simulaciones;index_simulacion++)
        array_fuerza_media[index_simulacion]=(float*)malloc(sizeof(float)*N_lineas);

    array_fuerza_error=(float**)malloc(sizeof(float*)*VARIABLE_punt->N_simulaciones);
    for(index_simulacion=0;index_simulacion<VARIABLE_punt->N_simulaciones;index_simulacion++)
        array_fuerza_error[index_simulacion]=(float*)malloc(sizeof(float)*N_lineas);

    array_pos=(float**)malloc(sizeof(float*)*N_lineas);
    for(i=0;i<N_lineas;i++)
        array_pos[i]=(float*)malloc(sizeof(float)*VARIABLE_punt->N_estadistica);

    array_pos_media=(float**)malloc(sizeof(float*)*VARIABLE_punt->N_simulaciones);
    for(index_simulacion=0;index_simulacion<VARIABLE_punt->N_simulaciones;index_simulacion++)
        array_pos_media[index_simulacion]=(float*)malloc(sizeof(float)*N_lineas);

    array_pos_error=(float**)malloc(sizeof(float*)*VARIABLE_punt->N_simulaciones);
    for(index_simulacion=0;index_simulacion<VARIABLE_punt->N_simulaciones;index_simulacion++)
        array_pos_error[index_simulacion]=(float*)malloc(sizeof(float)*N_lineas);


    for(index_simulacion=0;index_simulacion<VARIABLE_punt->N_simulaciones;index_simulacion++)
    {
        for(index_estad=0;index_estad<VARIABLE_punt->N_estadistica;index_estad++)
        {
            sprintf(read_fuerza_name,"pullf_%d_%d.xvg",index_simulacion,index_estad);
            read_fuerza=fopen(read_fuerza_name,"r");
            fseek(read_fuerza,puntero_pullf_basura[index_simulacion][index_estad],SEEK_SET);

            for(i=0;i<N_lineas;i++)
            {
                fscanf(read_fuerza,"%f %f",&xf,&yf);

                if(index_estad==0 && index_simulacion==0)
                    array_x[i]=xf;

                array_fuerza[i][index_estad]=yf;
            }
            fclose(read_fuerza);

            printf("Analisis trayectoria: %.3f %%\n",25.0+25.0*(index_simulacion*VARIABLE_punt->
                N_estadistica+index_estad+1.0)/(1.0*VARIABLE_punt->N_simulaciones*VARIABLE_punt->
                N_estadistica));
        }
        for(i=0;i<N_lineas;i++)
            MedVar(array_fuerza[i],VARIABLE_punt->N_estadistica,&array_fuerza_media[index_simulacion
                ][i],&array_fuerza_error[index_simulacion][i]);
    }

    for(index_simulacion=0;index_simulacion<VARIABLE_punt->N_simulaciones;index_simulacion++)
    {
        for(index_estad=0;index_estad<VARIABLE_punt->N_estadistica;index_estad++)
        {
            sprintf(read_pos_name,"pullx_%d_%d.xvg",index_simulacion,index_estad);
            read_pos=fopen(read_pos_name,"r");
            fseek(read_pos,puntero_pullx_basura[index_simulacion][index_estad],SEEK_SET);

            for(i=0;i<N_lineas;i++)
            {
                fscanf(read_pos,"%f %f",&xp,&yp);

                if(index_estad==0 && index_simulacion==0)
                    array_x[i]=xp;

                array_pos[i][index_estad]=yp;
            }
            fclose(read_pos);

            printf("Analisis trayectoria: %.3f %%\n",50.0+25.0*(index_simulacion*VARIABLE_punt->
                N_estadistica+index_estad+1.0)/(1.0*VARIABLE_punt->N_simulaciones*VARIABLE_punt->
                N_estadistica));
        }
        for(i=0;i<N_lineas;i++)
            MedVar(array_pos[i],VARIABLE_punt->N_estadistica,&array_pos_media[index_simulacion][i],&
                array_pos_error[index_simulacion][i]);
    }
```

```
for(i=0;i<VARIABLE_punt->N_simulaciones;i++)
    array_W[i]=0;

write_trayectoria=fopen("Trayectoria__Analisis.txt","w");
fprintf(write_trayectoria,"tiempo");
for(index_simulacion=0;index_simulacion<VARIABLE_punt->N_simulaciones;index_simulacion++)
    fprintf(write_trayectoria,"\tfuerza_media_%d\tposicion_%d\tW_%d",index_simulacion,
        index_simulacion,index_simulacion);
fprintf(write_trayectoria,"\n");
fclose(write_trayectoria);

for(i=0;i<N_lineas;i++)
{
    write_trayectoria=fopen("Trayectoria__Analisis.txt","a");
    fprintf(write_trayectoria,"%f",array_x[i]);
    for(index_simulacion=0;index_simulacion<VARIABLE_punt->N_simulaciones;index_simulacion++)
    {
        if(i==0)
            fprintf(write_trayectoria,"\t%f\t%f",array_fuerza_media[index_simulacion][i],
                array_pos_media[index_simulacion][i],0.0);
        if(i>0)
        {
            array_W[index_simulacion]+=(array_pos_media[index_simulacion][i]-array_pos_media[
                index_simulacion][i-1])*array_fuerza_media[index_simulacion][i];

            fprintf(write_trayectoria,"\t%f\t%f",array_fuerza_media[index_simulacion][i],
                array_pos_media[index_simulacion][i],array_W[index_simulacion]);
        }

        if(i%1000==0)
            printf("Analisis trayectoria: %.3f %%\n",75.0+25.0*(i*VARIABLE_punt->N_simulaciones+
                index_simulacion+1.0)/(1.0*N_lineas*VARIABLE_punt->N_simulaciones));
    }
    fprintf(write_trayectoria,"\n");
    fclose(write_trayectoria);
}



for(index_simulacion=0;index_simulacion<VARIABLE_punt->N_simulaciones;index_simulacion++)
{
    for(index_estad=0;index_estad<VARIABLE_punt->N_estadistica;index_estad++)
    {
        if(fork()==0)
        {
            sprintf(comando,"rm pullf_%d_%d.xvg pullx_%d_%d.xvg\n",index_simulacion,index_estad,
                index_simulacion,index_estad);

            pipe=popen(comando,"w");
            pclose(pipe);

            LiberarMemoriaReservada(CANAL_punt,POL_punt,MEMBRANA_punt,VARIABLE_punt); exit(0);
        }
        wait(NULL);
    }
}



estadistica_vel=(float*)malloc(sizeof(float)*VARIABLE_punt->N_estadistica);
for(i=0;i<VARIABLE_punt->N_estadistica;i++)
    estadistica_vel[i]=0;

estadistica_tiempo=(float*)malloc(sizeof(float)*VARIABLE_punt->N_estadistica);
for(i=0;i<VARIABLE_punt->N_estadistica;i++)
    estadistica_tiempo[i]=0;



sprintf(write_analisis_name,"PolPos=%.1f_CanalRadio=%.3f_tau=%.1f_pullMode=%s_E=%s__Analisis.txt
    ",POL_punt->posicion,CANAL_punt->radio,VARIABLE_punt->tau_t,VARIABLE_punt->pulling_modo,
    VARIABLE_punt->campo_E);

write_analisis=fopen(write_analisis_name,"w");
fprintf(write_analisis,"#SIM\tCANAL_bending_cte\tCANAL_bending_ang0\tCANAL_bond_k\tCANAL_bond_l0
    \tPOL_long_trans\tpull_rate[ps^-1]\tFrecuencia[ps^-1]\ttiempo_trans\terror_tiempo\
    tvelocidad_trans[ps^-1]\terror_vel\tfuerza_pull\terror_fuerza\n");
fclose(write_analisis);

printf("\n\n");
for(index_simulacion=0;index_simulacion<VARIABLE_punt->N_simulaciones;index_simulacion++)
{
    POL_long_trans=POL_punt->longitud[index_simulacion]-fabs((POL_punt->posicion+1)*0.5*POL_punt
        ->longitud[index_simulacion]+CANAL_punt->bond_reposo*0.5*CANAL_punt->Nb);

    for(index_estad=0;index_estad<VARIABLE_punt->N_estadistica;index_estad++)
    {
        fuerza_inst=0;
        fuerza_contador=0;
```

```
            sprintf(read_analisis_name,"TiempoTrans_%d_%d.txt",index_simulacion,index_estad);
            read_analisis=fopen(read_analisis_name,"r");
            fscanf(read_analisis,"%f",&tiempo_trans);
            fclose(read_analisis);


            estadistica_vel[index_estad]=POL_long_trans/(tiempo_trans*POL_punt->bond_reposo);
            estadistica_tiempo[index_estad]=tiempo_trans;


            sprintf(comando,"rm TiempoTrans_%d_%d.txt\n",index_simulacion,index_estad);

            pipe=popen(comando,"w");
            pclose(pipe);



            printf("Analisis global: %.3f %%\n",100.0*(index_simulacion*VARIABLE_punt->N_estadistica
                +index_estad+1.0)/(1.0*VARIABLE_punt->N_simulaciones*VARIABLE_punt->N_estadistica))
                ;

        }

        MedVar(estadistica_vel,VARIABLE_punt->N_estadistica,&vel_medio,&error_vel);
        MedVar(estadistica_tiempo,VARIABLE_punt->N_estadistica,&tiempo_medio,&error_tiempo);
        MedVar(array_fuerza_media[index_simulacion],N_lineas,&fuerza_medio,&error_fuerza);

        write_analisis=fopen(write_analisis_name,"a");
        if(Igual(tiempo_trans,0)==1)
            fprintf(write_analisis,"%d\t%f\t%f\t%f\t%f\t%s %s\n",index_simulacion, CANAL_punt->
                bending_fuerza[index_simulacion],CANAL_punt->bending_ang,CANAL_punt->bond_fuerza[
                index_simulacion],CANAL_punt->bond_reposo,POL_punt->longitud[index_simulacion],"NO"
                ,"TRASLOCACION");
        else
            fprintf(write_analisis,"%d\t%f\t%f\t%f\t%f\t%f\t%f\t%f\t%f\t%f\t%f\t%f\t%f\n",
                index_simulacion, CANAL_punt->bending_fuerza[index_simulacion],CANAL_punt->
                bending_ang,CANAL_punt->bond_fuerza[index_simulacion],CANAL_punt->bond_reposo,
                POL_long_trans,VARIABLE_punt->pull_rate[index_simulacion]/POL_punt->bond_reposo,
                VARIABLE_punt->Ezfrecuencia[index_simulacion],tiempo_medio,error_tiempo,vel_medio,
                error_vel,fuerza_medio,error_fuerza);
        fclose(write_analisis);

    }



    free(array_x);

    free(array_W);

    for(i=0;i<N_lineas;i++)
        free(array_fuerza[i]);
    free(array_fuerza);


    for(index_simulacion=0;index_simulacion<VARIABLE_punt->N_simulaciones;index_simulacion++)
        free(array_fuerza_error[index_simulacion]);
    free(array_fuerza_error);

    for(i=0;i<N_lineas;i++)
        free(array_pos[i]);
    free(array_pos);

    for(index_simulacion=0;index_simulacion<VARIABLE_punt->N_simulaciones;index_simulacion++)
        free(array_pos_media[index_simulacion]);
    free(array_pos_media);

    for(index_simulacion=0;index_simulacion<VARIABLE_punt->N_simulaciones;index_simulacion++)
        free(array_pos_error[index_simulacion]);
    free(array_pos_error);

    for(index_simulacion=0;index_simulacion<VARIABLE_punt->N_simulaciones;index_simulacion++)
        free(puntero_pullf_basura[index_simulacion]);
    free(puntero_pullf_basura);

    for(index_simulacion=0;index_simulacion<VARIABLE_punt->N_simulaciones;index_simulacion++)
        free(puntero_pullx_basura[index_simulacion]);
    free(puntero_pullx_basura);

    free(estadistica_vel);
    free(estadistica_tiempo);

    for(index_simulacion=0;index_simulacion<VARIABLE_punt->N_simulaciones;index_simulacion++)
        free(array_fuerza_media[index_simulacion]);
    free(array_fuerza_media);
}



///LEER INPUT
    ////////////////////////////////////////////////////////////////////////////////////////////
```

```
void LeerInput(struct CANAL_struct* CANAL_punt, struct POL_struct* POL_punt, struct MEMBRANA_struct*
     MEMBRANA_punt, struct VARIABLE_struct* VARIABLE_punt)
{
    int i;
    FILE *input;
    FILE *min,*term,*water,*martini,*dry_martini, *Lterm;
    char Pcoupl[256];
    float ref_p;
    char input_string[256];
    char basura[256];

    input=fopen("./Input/genconf.txt","r");

    fscanf(input,"%s %d\n",basura,basura,&VARIABLE_punt->N_simulaciones);

    CANAL_punt->bond_fuerza=(float*)malloc(sizeof(float)*(VARIABLE_punt->N_simulaciones));
    CANAL_punt->bending_fuerza=(float*)malloc(sizeof(float)*(VARIABLE_punt->N_simulaciones));

    VARIABLE_punt->pull_rate=(float*)malloc(sizeof(float)*(VARIABLE_punt->N_simulaciones));
    VARIABLE_punt->pull_fuerza=(float*)malloc(sizeof(float)*(VARIABLE_punt->N_simulaciones));
    VARIABLE_punt->Ezfrecuencia=(float*)malloc(sizeof(float)*(VARIABLE_punt->N_simulaciones));
    POL_punt->longitud=(float*)malloc(sizeof(float)*(VARIABLE_punt->N_simulaciones));

    fscanf(input,"%s %s %d\n",basura,basura,&VARIABLE_punt->N_estadistica);

    fscanf(input,"%s %s %f\n",basura,basura,&VARIABLE_punt->term_tol_derivada);
    fscanf(input,"%s %s %f\n",basura,basura,&VARIABLE_punt->term_tolerancia);
    fscanf(input,"%s %s %d\n",basura,basura,&VARIABLE_punt->N_term);
    fscanf(input,"%s %s %d\n",basura,basura,&VARIABLE_punt->N_Lterm);
    fscanf(input,"%s %s %f\n",basura,basura,&VARIABLE_punt->N_sim_cte);

    fscanf(input,"%s\t%s\t%s\n",basura,basura,VARIABLE_punt->PintaTrayectoria);
    fscanf(input,"%s\t%s\t%s\n",basura,basura,VARIABLE_punt->conf_externa);

    fscanf(input,"%s\t%s\t%s\n",basura,basura,VARIABLE_punt->Acoplo_presion);

    fscanf(input,"%s\t%s\t%s\n",basura,basura,CANAL_punt->Borde_poro);
    fscanf(input,"%s\t%s\t%s\n",basura,basura,CANAL_punt->Cuerpo_poro);
    fscanf(input,"%s\t%s\t%s\n",basura,basura,POL_punt->Borde_polimero);
    fscanf(input,"%s\t%s\t%s\n",basura,basura,POL_punt->Cuerpo_polimero);

    fscanf(input,"%s\t%s\t%s\n",basura,basura,VARIABLE_punt->pulling);
    fscanf(input,"%s\t%s\t%s\n",basura,basura,VARIABLE_punt->pulling_modo);

    fscanf(input,"%s %s",basura,basura);
    for(i=0;i<VARIABLE_punt->N_simulaciones;i++)
        fscanf(input,"%f",VARIABLE_punt->pull_rate+i);

    fscanf(input,"%s %s",basura,basura);
    for(i=0;i<VARIABLE_punt->N_simulaciones;i++)
        fscanf(input,"%f",VARIABLE_punt->pull_fuerza+i);

    fscanf(input,"%s %s %f",basura,basura,&VARIABLE_punt->pull_k);
    fscanf(input,"%s %s %f",basura,basura,&VARIABLE_punt->ref_t);
    fscanf(input,"%s %s %f",basura,basura,&VARIABLE_punt->tau_t);

    fscanf(input,"%s\t%s\t%s\n",basura,basura,input_string);
    if(strcmp(input_string,"si")==0)
        VARIABLE_punt->AGUA_sel=1;
    else
        VARIABLE_punt->AGUA_sel=0;


    fscanf(input,"%s %s %s",basura,basura,input_string);
    if(strcmp(input_string,"si")==0)
        CANAL_punt->sel=1;
    else
        CANAL_punt->sel=0;

    fscanf(input,"%s %s %f",basura,basura,&CANAL_punt->radio);

    fscanf(input,"%s %s %f",basura,basura,&CANAL_punt->longitudInterior);


    fscanf(input,"%s %s %f",basura,basura,&CANAL_punt->longitudExterior);


    fscanf(input,"%s %s",basura,basura);
    for(i=0;i<VARIABLE_punt->N_simulaciones;i++)
        fscanf(input,"%f",CANAL_punt->bond_fuerza+i);


    fscanf(input,"%s %s %f",basura,basura,&CANAL_punt->bond_reposo);


    fscanf(input,"%s %s",basura,basura);
    for(i=0;i<VARIABLE_punt->N_simulaciones;i++)
        fscanf(input,"%f",CANAL_punt->bending_fuerza+i);


    fscanf(input,"%s %s %f",basura,basura,&CANAL_punt->bending_ang);
```

```
fscanf(input,"%s %s %s",basura,basura,input_string);
if(strcmp(input_string,"si")==0)
    POL_punt->sel=1;
else
    POL_punt->sel=0;

fscanf(input,"%s %s %f",basura,basura,&POL_punt->bond_fuerza);


fscanf(input,"%s %s %f",basura,basura,&POL_punt->bond_reposo);


fscanf(input,"%s %s %f",basura,basura,&POL_punt->bending_fuerza);


fscanf(input,"%s %s %f",basura,basura,&POL_punt->bending_ang);

fscanf(input,"%s %s",basura,basura);
for(i=0;i<VARIABLE_punt->N_simulaciones;i++)
    fscanf(input,"%f",POL_punt->longitud+i);


fscanf(input,"%s %s %f",basura,basura,&POL_punt->posicion);


fscanf(input,"%s %s %s",basura,basura,input_string);
if(strcmp(input_string,"si")==0)
    MEMBRANA_punt->sel=1;
else
    MEMBRANA_punt->sel=0;


fscanf(input,"%s %s %f",basura,basura,&MEMBRANA_punt->area);

fscanf(input,"%s\t%s\t%s\n",basura,basura,VARIABLE_punt->campo_E);
fscanf(input,"%s %s",basura,basura);
for(i=0;i<VARIABLE_punt->N_simulaciones;i++)
    fscanf(input,"%f",VARIABLE_punt->Ezfrecuencia+i);

fscanf(input,"%s %s %f\n",basura,basura,&VARIABLE_punt->Ezintensidad);
fscanf(input,"%s %s %f\n",basura,basura,&VARIABLE_punt->Exintensidad);
fscanf(input,"%s %s %f\n",basura,basura,&VARIABLE_punt->Eyintensidad);


fclose(input);


for(i=0;i<VARIABLE_punt->N_simulaciones;i++)
{
    CANAL_punt->Nb=(int)((CANAL_punt->longitudInterior+2*CANAL_punt->longitudExterior)/
        CANAL_punt->bond_reposo);
    CANAL_punt->NExterior=(int)(CANAL_punt->longitudExterior/CANAL_punt->bond_reposo);
    CANAL_punt->phi=acos(1-CANAL_punt->bond_reposo*CANAL_punt->bond_reposo/(2*CANAL_punt->radio*
        CANAL_punt->radio));
    CANAL_punt->M=(int)(2*PI/CANAL_punt->phi);
    CANAL_punt->n=CANAL_punt->Nb*CANAL_punt->M;
}


min=fopen("Input/minim.mdp","w");
term=fopen("Input/term.mdp","w");
Lterm=fopen("Input/Lterm.mdp","w");
water=fopen("Input/water.gro","w");
martini=fopen("Input/martini_v2.2.itp","w");
dry_martini=fopen("Input/dry_martini_v2.1.itp","w");

if(strcmp(VARIABLE_punt->Acoplo_presion,"si")==0)
{
    sprintf(Pcoupl,"parrinello-rahman");

    if(strcmp(VARIABLE_punt->pulling,"si")==0)
        ref_p=0.5;

    if(strcmp(VARIABLE_punt->pulling,"no")==0)
        ref_p=1.0;
}
else
{
    sprintf(Pcoupl,"no");
    ref_p=1.0;
}

fprintf(min,"integrator\t= steep\n\nemstep\t\t = 0.001\nemtol\t\t= 1000.0\nnsteps\t\t= 5000\
    ncomm-mode\t= Linear\ncomm-grps\t= \n\nnstlist\t\t = 1\nnstxout\t\t = 0\nnstvout\t\t = 0\
    nnstfout\t\t = 0\nnstlog\t\t = 0\nnstenergy\t\t = 0\nnstxout-compressed\t\t = -1\ncutoff-
    scheme\t\t = verlet \nverlet-buffer-drift\t= 0.005\nns_type\t\t = grid\n\ncoulombtype\t\t =
     reaction-field\n\nrcoulomb\t\t = 1.2\n\nvdw-type\t\t = cutoff\nvdw-modifier\t\t =
    Potential-shift-verlet\nrvdw\t\t = 1.2\n\npbc\t\t = xyz\n");


fprintf(term,"\ntitle\t\t = Termalizacion\n\n\n\nintegrator\t\t = sd\ndt\t\t = 0.01\nnsteps\t\
    t = %d\nstcomm\t\t = 100\ncomm-mode\t\t = Linear\ncomm-grps\t\t = \n\n\nstxout\t\t = %d
    \nstvout\t\t = %d\nstfout\t\t = %d\nstlog\t\t = 1000\nstenergy\t\t = 1000\nstxout-
```

```
            compressed\t\t = -1\ncompressed-x-grps\t\t = SECO\n\nfreezegrps\t\t = DENTRO CHNNL\
            nfreezedim\t\t = Y Y Y N N N Y\n\ncutoff-scheme\t\t = Verlet\nnstlist\t\t = 10\nns_type\t\t =
             grid\npbc\t\t = xyz\nverlet-buffer-tolerance\t = 0.005\n\n\ncoulombtype\t\t = reaction-
            field\nrcoulomb\t\t = 1.1\nepsilon_r\t\t = 15\t\n\nvdw_type\t\t = cutoff \nvdw-modifier\t\t
             = Potential-shift-verlet\nrvdw\t\t = 1.1\t\n\n\n\ntcoupl\t\t = v-rescale\ntc-grps\t\t =
            SECO\ntau_t\t\t = %.1f\nref_t\t\t = %.2f\nPcoupl\t\t = %s\nPcoupltype\t\t = semiisotropic\
            ntau_p\t\t = 12.0\ncompressibility\t\t = 3e-4 3e-4\nref_p\t\t = %.2f %.2f\n\ngen_vel\t\t =
            no \ngen_temp\t\t = 320 \ngen_seed\t\t = 473529\n\n\nconstraints\t\t = none \
            nconstraint_algorithm\t\t = Lincs\nperiodic-molecules = yes\n",VARIABLE_punt->N_term,
            VARIABLE_punt->N_term,VARIABLE_punt->N_term,VARIABLE_punt->N_term,VARIABLE_punt->tau_t,
            VARIABLE_punt->ref_t,Pcoupl,ref_p,ref_p);

    fprintf(Lterm,"\ntitle\t\t = LTermalizacion\n\n\n\nintegrator\t\t = sd\ndt\t\t = 0.01\nnsteps\
        t\t = %d\nstcomm\t\t = 100\ncomm-mode\t\t = Linear\ncomm-grps\t\t = \n\n\nstxout\t\t =
        %d\nstvout\t\t = %d\nstfout\t\t = %d\nstlog\t\t = 1000\nstenergy\t\t = 1000\nstxout-
        compressed\t\t = -1\ncompressed-x-grps\t\t = SECO\n\nfreezegrps\t\t = DENTRO CHNNL\
        nfreezedim\t\t = Y Y Y N N N Y\n\ncutoff-scheme\t\t = Verlet\nstlist\t\t = 10\ns_type\t\t =
         grid\npbc\t\t = xyz\nverlet-buffer-tolerance\t = 0.005\n\n\ncoulombtype\t\t = reaction-
        field\nrcoulomb\t\t = 1.1\nepsilon_r\t\t = 15\t\n\nvdw_type\t\t = cutoff \nvdw-modifier\t\t
         = Potential-shift-verlet\nrvdw\t\t = 1.1\t\n\n\n\ntcoupl\t\t = v-rescale\ntc-grps\t\t =
        SISTEMA\ntau_t\t\t = %.1f\nref_t\t\t = %.2f\nPcoupl\t\t = %s\nPcoupltype\t\t =
        semiisotropic\ntau_p\t\t = 12.0\ncompressibility\t\t = 3e-4 3e-4\nref_p\t\t = %.2f %.2f\n\
        ngen_vel\t\t = no \ngen_temp\t\t = 320 \ngen_seed\t\t = 473529\n\n\nconstraints\t\t = none
        \nconstraint_algorithm\t\t = Lincs\nperiodic-molecules = yes\n",VARIABLE_punt->N_Lterm,
        VARIABLE_punt->N_Lterm,VARIABLE_punt->N_Lterm,VARIABLE_punt->N_Lterm,VARIABLE_punt->tau_t,
        VARIABLE_punt->ref_t,Pcoupl,ref_p,ref_p);


    fprintf(water,"[water.gro]"");


    fprintf(martini,"[martini.itp]");

    fprintf(dry_martini,"[dry_martini.itp]");

    fclose(min);
    fclose(term);
    fclose(Lterm);
    fclose(water);
    fclose(martini);
    fclose(dry_martini);
}



///LIBERACION DE MEMORIA Y LIMPIEZA////////////////////////////////////
void LiberarMemoriaReservada(struct CANAL_struct* CANAL_punt,struct POL_struct* POL_punt, struct
     MEMBRANA_struct* MEMBRANA_punt,struct VARIABLE_struct* VARIABLE_punt)
{
    int i;

    free(CANAL_punt->bond_fuerza);
    free(CANAL_punt->bending_fuerza);
    free(VARIABLE_punt->pull_rate);
    free(VARIABLE_punt->pull_fuerza);

    for(i=0;i<VARIABLE_punt->N_simulaciones;i++)
        free(VARIABLE_punt->MatrizSeguridad[i]);
    free(VARIABLE_punt->MatrizSeguridad);

    free(VARIABLE_punt->ArraySeguridad);

    free(VARIABLE_punt->Ezfrecuencia);

    free(POL_punt->longitud);

}

void LimpiaEstructurasUsadas(struct CANAL_struct* CANAL_punt,struct POL_struct* POL_punt, struct
     MEMBRANA_struct* MEMBRANA_punt,struct VARIABLE_struct* VARIABLE_punt,int index_simulacion)
{
    char comando[8192];
    FILE *pipe;
    if(fork()==0)
    {
        sprintf(comando,"rm SISTEMA_canal_%d.itp SISTEMA_dspc_%d.itp SISTEMA_pol_%d.itp topol_%d.top
              Sistema_term_%d.gro SISTEMA_index_%d.ndx\n",index_simulacion,index_simulacion,
             index_simulacion,index_simulacion,index_simulacion,index_simulacion);

        pipe=popen(comando,"w");
        pclose(pipe);

        LiberarMemoriaReservada(CANAL_punt,POL_punt,MEMBRANA_punt,VARIABLE_punt); exit(0);
    }
    wait(NULL);
}
```

```
/*************************
 *                       *
 * FUNCIONES ESPECIFICAS *
 *                       *
 *************************/

void GeneraMDP(struct CANAL_struct* CANAL_punt,struct POL_struct* POL_punt,struct VARIABLE_struct*
     VARIABLE_punt, int index_simulacion,int index_estad)
{
     FILE *sim;
     char Pcoupl[256];
     char pull_geometry[256];
     char martini_md_name[256];
     float ref_p;
     int nstxout,npullout;

     if(strcmp(VARIABLE_punt->pulling_modo,"velocidad")==0)
          VARIABLE_punt->N_sim=(int)(VARIABLE_punt->N_sim_cte*(POL_punt->longitud[index_simulacion]+
               CANAL_punt->longitudExterior+CANAL_punt->longitudInterior)/(0.01*VARIABLE_punt->
               pull_rate[index_simulacion]));

     if(strcmp(VARIABLE_punt->pulling_modo,"fuerza")==0)
          VARIABLE_punt->N_sim=(int)(VARIABLE_punt->N_sim_cte*((POL_punt->longitud[index_simulacion]+
               CANAL_punt->longitudExterior+CANAL_punt->longitudInterior)*72.0*((int)(POL_punt->
               longitud[index_simulacion]/POL_punt->bond_reposo)))/(0.01*VARIABLE_punt->pull_fuerza[
               index_simulacion]*VARIABLE_punt->tau_t));



     if(strcmp(VARIABLE_punt->Acoplo_presion,"si")==0)
     {
          sprintf(Pcoupl,"parrinello-rahman");
          sprintf(pull_geometry,"direction");

          if(strcmp(VARIABLE_punt->pulling,"si")==0)
               ref_p=0.5;

          if(strcmp(VARIABLE_punt->pulling,"no")==0)
               ref_p=1.0;
     }
     else
     {
          sprintf(Pcoupl,"no");
          sprintf(pull_geometry,"direction-periodic");
          ref_p=1.0;
     }

     if(strcmp(VARIABLE_punt->PintaTrayectoria,"si")==0)
          nstxout=VARIABLE_punt->N_sim/1000;
     else
          nstxout=10;

     if(VARIABLE_punt->N_sim<35000)
          npullout=1;
     else
          npullout=1+VARIABLE_punt->N_sim/35000;

     sprintf(martini_md_name,"Input/martini_md_%d_%d.mdp",index_simulacion,index_estad);

     sim=fopen(martini_md_name,"w");



     if(strcmp(VARIABLE_punt->pulling,"si")==0 && strcmp(VARIABLE_punt->campo_E,"si")==0)
     {
          if(strcmp(VARIABLE_punt->pulling_modo,"velocidad")==0)
               fprintf(sim,"\ntitle\t\t = Simulacion (Modo = Pull)\n\n\n\nintegrator\t\t = sd\ndt\t\t
                    = 0.01   \nnsteps\t\t = %d\nnstcomm\t\t = 100\ncomm-mode\t\t = Linear\ncomm-grps\t\
                    t = \n\n\nnstxout\t\t = %d\nstvout\t\t = %d\nstfout\t\t = %d\nstlog\t\t = %d\
                    \nstenergy\t\t = 1000\nstxout-compressed\t\t = -1\ncompressed-x-grps\t\t = SECO\n\
                    \ncutoff-scheme\t\t = Verlet\nstlist\t\t = 10\ns_type\t\t = grid\npbc\t\t = xyz\
                    \nverlet-buffer-tolerance\t = 0.005\n\ncoulombtype\t\t = reaction-field\rcoulomb\
                    t\t = 1.1\epsilon_r\t\t = 150\t\n\nvdw_type\t\t = cutoff \nvdw-modifier\t\t =
                    Potential-shift-verlet\nrvdw\t\t = 1.1\t\n\n\ntcoupl\t\t = v-rescale\ntc-grps\t\t
                     = SISTEMA\ntau_t\t\t = %.1f\nref_t\t\t = %.1f\nPcoupl\t\t = %s\nPcoupltype\t\t =
                    semiisotropic  \ntau_p\t\t = 12.0\ncompressibility\t\t = 3e-4 3e-4\nref_p\t\t = %.2
                    f %.2f\n\ngen_vel\t\t = no \ngen_temp\t\t = 320 \ngen_seed\t\t = 473529\n\npull\t\t
                    \t = yes\npull-ngroups\t\t = 2\npull-ncoords\t\t = 1\npull-coord1-groups\t = 1 2 \
                    npull-coord1-geometry\t = %s\npull-group1-name\t = CONTRAPESO\npull-group2-name\t =
                     PULLING\npull-coord1-type\t = umbrella\npull-coord1-vec\t\t = 0.0 0.0 1.0\npull-
                    coord1-start\t = yes\npull-coord1-rate\t = %f\npull-coord1-k\t\t = %.1f\npull-
                    nstxout\t = %d\npull-nstfout\t = %d\n\n \n\n\nconstraints\t\t = none \
                    nconstraint_algorithm\t\t = Lincs\nE-z = 1  %.2f 0\nE-x = 1  %.2f 0\nE-y = 1  %.2f 0\
                    nE-zt = 1  %.2f 0\nE-xt = 1  %.2f 0\nE-yt = 1  %.2f 0\nperiodic-molecules = yes\n",
                    VARIABLE_punt->N_sim,nstxout,VARIABLE_punt->N_sim,VARIABLE_punt->N_sim,
                    VARIABLE_punt->N_sim,VARIABLE_punt->tau_t,VARIABLE_punt->ref_t,Pcoupl,ref_p,ref_p,
                    pull_geometry,VARIABLE_punt->pull_rate[index_simulacion],VARIABLE_punt->pull_k,
```

```
                npullout,npullout,VARIABLE_punt->Ezintensidad,VARIABLE_punt->Exintensidad,
                VARIABLE_punt->Eyintensidad,2*PI*VARIABLE_punt->Ezfrecuencia[index_simulacion],2*PI
                *VARIABLE_punt->Ezfrecuencia[index_simulacion],2*PI*VARIABLE_punt->Ezfrecuencia[
                index_simulacion]);

        if(strcmp(VARIABLE_punt->pulling_modo,"fuerza")==0 && strcmp(VARIABLE_punt->campo_E,"si")
            ==0)
            fprintf(sim,"\ntitle\t\t = Simulacion (Modo = Pull)\n\n\n\nintegrator\t\t = sd\ndt\t\t
                 = 0.01  \nnsteps\t\t = %d\nstcomm\t\t = 100\ncomm-mode\t\t = Linear\ncomm-grps\t\
                t = \n\n\n\nstxout\t\t = %d\nstvout\t\t = %d\nstfout\t\t = %d\nstlog\t\t =
                1000\nstenergy\t\t = 1000\nstxout-compressed\t\t = -1\ncompressed-x-grps\t\t =
                SECO\n\n\ncutoff-scheme\t\t = Verlet\nstlist\t\t = 10\ns_type\t\t = grid\npbc\t\t
                 = xyz\nverlet-buffer-tolerance\t = 0.005\n\n\ncoulombtype\t\t = reaction-field\
                nrcoulomb\t\t = 1.1\nepsilon_r\t\t = 150\t\n\nvdw_type\t\t = cutoff \nvdw-modifier\
                t\t = Potential-shift-verlet\nrvdw\t\t = 1.1\t\n\n\ntcoupl\t\t = v-rescale\ntc-
                grps\t\t = SISTEMA\ntau_t\t\t = %.1f\nref_t\t\t = %.1f\nPcoupl\t\t = %s\nPcoupltype
                \t\t = semiisotropic  \ntau_p\t\t = 12.0\ncompressibility\t\t = 3e-4 3e-4\nref_p\t\
                t = %.2f %.2f\n\ngen_vel\t\t = no \ngen_temp\t\t = 320 \ngen_seed\t\t = 473529\n\
                npull\t\t\t = yes\npull-ngroups\t\t = 2\npull-ncoords\t\t = 1\npull-coord1-groups\t
                 = 1 2 \npull-coord1-geometry\t = %s\npull-group1-name\t = CONTRAPESO\npull-group2-
                name\t = PULLING\npull-coord1-type\t = constant-force\npull-coord1-vec\t\t = 0.0
                0.0 1.0\npull-coord1-start\t = yes\npull-coord1-k\t\t = %f\npull-nstxout\t = %d\
                npull-nstfout\t = %d\n\n \n\n\nconstraints\t\t = none \nconstraint_algorithm\t\t =
                Lincs\nE-z = 1 %.2f 0\nE-x = 1 %.2f 0\nE-y = 1 %.2f 0\nE-zt = 1 %.2f 0\nE-xt = 1
                %.2f 0\nE-yt = 1 %.2f 0\nperiodic-molecules = yes\n",VARIABLE_punt->N_sim,nstxout,
                VARIABLE_punt->N_sim,VARIABLE_punt->N_sim,VARIABLE_punt->tau_t,VARIABLE_punt->ref_t
                ,Pcoupl,ref_p,ref_p,pull_geometry,-1*VARIABLE_punt->pull_fuerza[index_simulacion],
                npullout,npullout,VARIABLE_punt->Ezintensidad,VARIABLE_punt->Exintensidad,
                VARIABLE_punt->Eyintensidad,2*PI*VARIABLE_punt->Ezfrecuencia[index_simulacion],2*PI
                *VARIABLE_punt->Ezfrecuencia[index_simulacion],2*PI*VARIABLE_punt->Ezfrecuencia[
                index_simulacion]);
}

if(strcmp(VARIABLE_punt->pulling,"no")==0 && strcmp(VARIABLE_punt->campo_E,"si")==0)
    fprintf(sim,"\ntitle\t\t = Simulacion (Modo = Difusion)\n\n\n\nintegrator\t\t = sd\ndt\t\t
         = 0.01\nnsteps\t\t = %d\nstcomm\t\t = 100\ncomm-mode\t\t = Linear\ncomm-grps\t\t = \n
        \n\n\nstxout\t\t = %d\nstvout\t\t = %d\nstfout\t\t = %d\nstlog\t\t = 1000\
        nstenergy\t\t = 1000\nstxout-compressed\t\t = -1\ncompressed-x-grps\t\t = SECO\n\n\
        ncutoff-scheme\t\t = Verlet\nstlist\t\t = 10\ns_type\t\t = grid\npbc\t\t = xyz\
        nverlet-buffer-tolerance\t = 0.005\n\n\ncoulombtype\t\t = reaction-field\nrcoulomb\t\t
        = 1.1\nepsilon_r\t\t = 150\t\n\nvdw_type\t\t = cutoff \nvdw-modifier\t\t = Potential-
        shift-verlet\nrvdw\t\t = 1.1\t\n\n\ntcoupl\t\t = v-rescale\ntc-grps\t\t = SISTEMA\
        ntau_t\t\t = %.1f\nref_t\t\t = %.1f\nPcoupl\t\t = %s\nPcoupltype\t\t = semiisotropic\
        ntau_p\t\t = 12.0\ncompressibility\t\t = 3e-4 3e-4\nref_p\t\t = %.2f %.2f\n\ngen_vel\t\t
        = no \ngen_temp\t\t = 320 \ngen_seed\t\t = 473529\n\n\nconstraints\t\t = none \
        nconstraint_algorithm\t\t = Lincs\nE-z = 1 %.2f 0\nE-x = 1 %.2f 0\nE-y = 1 %.2f 0\nE-zt
         = 1 %.2f 0\nE-xt = 1 %.2f 0\nE-yt = 1 %.2f 0\nperiodic-molecules = yes\n",
        VARIABLE_punt->N_sim,nstxout,VARIABLE_punt->N_sim,VARIABLE_punt->N_sim,VARIABLE_punt->
        tau_t,VARIABLE_punt->ref_t,Pcoupl,ref_p,ref_p,VARIABLE_punt->Ezintensidad,VARIABLE_punt
        ->Exintensidad,VARIABLE_punt->Eyintensidad,2*PI*VARIABLE_punt->Ezfrecuencia[
        index_simulacion],2*PI*VARIABLE_punt->Ezfrecuencia[index_simulacion],2*PI*VARIABLE_punt
        ->Ezfrecuencia[index_simulacion]);



if(strcmp(VARIABLE_punt->pulling,"si")==0 && strcmp(VARIABLE_punt->campo_E,"no")==0)
{
    if(strcmp(VARIABLE_punt->pulling_modo,"velocidad")==0)
        fprintf(sim,"\ntitle\t\t = Simulacion (Modo = Pull)\n\n\n\nintegrator\t\t = sd\ndt\t\t
             = 0.01  \nnsteps\t\t = %d\nstcomm\t\t = 100\ncomm-mode\t\t = Linear\ncomm-grps\t\
            t = \n\n\n\nstxout\t\t = %d\nstvout\t\t = %d\nstfout\t\t = %d\nstlog\t\t = %d\
            nstenergy\t\t = 1000\nstxout-compressed\t\t = -1\ncompressed-x-grps\t\t = SECO\n\
            n\ncutoff-scheme\t\t = Verlet\nstlist\t\t = 10\ns_type\t\t = grid\npbc\t\t = xyz\
            nverlet-buffer-tolerance\t = 0.005\n\n\ncoulombtype\t\t = reaction-field\nrcoulomb\
            t\t = 1.1\nepsilon_r\t\t = 15\t\n\nvdw_type\t\t = cutoff \nvdw-modifier\t\t =
            Potential-shift-verlet\nrvdw\t\t = 1.1\t\n\n\ntcoupl\t\t = v-rescale\ntc-grps\t\t
             = SISTEMA\ntau_t\t\t = %.1f\nref_t\t\t = %.1f\nPcoupl\t\t = %s\nPcoupltype\t\t =
            semiisotropic  \ntau_p\t\t = 12.0\ncompressibility\t\t = 3e-4 3e-4\nref_p\t\t = %.2
            f %.2f\n\ngen_vel\t\t = no \ngen_temp\t\t = 320 \ngen_seed\t\t = 473529\n\npull\t\t
            \t = yes\npull-ngroups\t\t = 2\npull-ncoords\t\t = 1\npull-coord1-groups\t = 1 2 \
            npull-coord1-geometry\t = %s\npull-group1-name\t = CONTRAPESO\npull-group2-name\t =
             PULLING\npull-coord1-type\t = umbrella\npull-coord1-vec\t\t = 0.0 0.0 1.0\npull-
            coord1-start\t = yes\npull-coord1-rate\t = %f\npull-coord1-k\t\t = %.1f\npull-
            nstxout\t = %d\npull-nstfout\t = %d\n\n \n\n\nconstraints\t\t = none \
            nconstraint_algorithm\t\t = Lincs\nperiodic-molecules = yes\n",VARIABLE_punt->N_sim
            ,nstxout,VARIABLE_punt->N_sim,VARIABLE_punt->N_sim,VARIABLE_punt->N_sim,
            VARIABLE_punt->tau_t,VARIABLE_punt->ref_t,Pcoupl,ref_p,ref_p,pull_geometry,
            VARIABLE_punt->pull_rate[index_simulacion],VARIABLE_punt->pull_k,npullout,npullout)
            ;

    if(strcmp(VARIABLE_punt->pulling_modo,"fuerza")==0 && strcmp(VARIABLE_punt->campo_E,"no")
        ==0)
        fprintf(sim,"\ntitle\t\t = Simulacion (Modo = Pull)\n\n\n\nintegrator\t\t = sd\ndt\t\t
             = 0.01  \nnsteps\t\t = %d\nstcomm\t\t = 100\ncomm-mode\t\t = Linear\ncomm-grps\t\
            t = \n\n\n\nstxout\t\t = %d\nstvout\t\t = %d\nstfout\t\t = %d\nstlog\t\t =
            1000\nstenergy\t\t = 1000\nstxout-compressed\t\t = -1\ncompressed-x-grps\t\t =
            SECO\n\n\ncutoff-scheme\t\t = Verlet\nstlist\t\t = 10\ns_type\t\t = grid\npbc\t\t
             = xyz\nverlet-buffer-tolerance\t = 0.005\n\n\ncoulombtype\t\t = reaction-field\
            nrcoulomb\t\t = 1.1\nepsilon_r\t\t = 15\t\n\nvdw_type\t\t = cutoff \nvdw-modifier\t
            \t = Potential-shift-verlet\nrvdw\t\t = 1.1\t\n\n\ntcoupl\t\t = v-rescale\ntc-
            grps\t\t = SISTEMA\ntau_t\t\t = %.1f\nref_t\t\t = %.1f\nPcoupl\t\t = %s\nPcoupltype
            \t\t = semiisotropic  \ntau_p\t\t = 12.0\ncompressibility\t\t = 3e-4 3e-4\nref_p\t\
```

```
                    t = %.2f %.2f\n\ngen_vel\t\t = no \ngen_temp\t\t = 320 \ngen_seed\t\t = 473529\n\
                    npull\t\t\t = yes\npull-ngroups\t\t = 2\npull-ncoords\t\t = 1\npull-coord1-groups\t
                     = 1 2 \npull-coord1-geometry\t = %s\npull-group1-name\t = CONTRAPESO\npull-group2-
                    name\t = PULLING\npull-coord1-type\t = constant-force\npull-coord1-vec\t\t = 0.0
                    0.0 1.0\npull-coord1-start\t = yes\npull-coord1-k\t\t = %f\npull-nstxout\t = %d\
                    npull-nstfout\t = %d\n\n \n\nconstraints\t\t = none \nconstraint_algorithm\t\t =
                    Lincs\nperiodic-molecules = yes\n",VARIABLE_punt->N_sim,nstxout,VARIABLE_punt->
                    N_sim,VARIABLE_punt->N_sim,VARIABLE_punt->tau_t,VARIABLE_punt->ref_t,Pcoupl,ref_p,
                    ref_p,pull_geometry,-1*VARIABLE_punt->pull_fuerza[index_simulacion],npullout,
                    npullout);
    }

    if(strcmp(VARIABLE_punt->pulling,"no")==0 && strcmp(VARIABLE_punt->campo_E,"no")==0)
        fprintf(sim,"\ntitle\t\t = Simulacion (Modo = Difusion)\n\n\n\nintegrator\t\t = sd\ndt\t\t
              = 0.01\nsteps\t\t = %d\nstcomm\t\t = 100\ncomm-mode\t\t = Linear\ncomm-grps\t\t = \n
            \n\n\nstxout\t\t = %d\nstvout\t\t = %d\nstfout\t\t = %d\nstlog\t\t = 1000\
            nstenergy\t\t = 1000\nstxout-compressed\t\t = -1\ncompressed-x-grps\t\t = SECO\n\n\
            ncutoff-scheme\t\t = Verlet\nstlist\t\t = 10\ns_type\t\t = grid\npbc\t\t = xyz\
            nverlet-buffer-tolerance\t = 0.005\n\ncoulombtype\t\t = reaction-field\rcoulomb\t\t
            = 1.1\epsilon_r\t\t = 15\t\n\vdw_type\t\t = cutoff \vdw-modifier\t\t = Potential-
            shift-verlet\rvdw\t\t = 1.1\t\n\n\tcoupl\t\t = v-rescale\tc-grps\t\t = SISTEMA\
            tau_t\t\t = %.1f\ref_t\t\t = %.1f\nPcoupl\t\t = %s\nPcoupltype\t\t = semiisotropic\
            tau_p\t\t = 12.0\compressibility\t\t = 3e-4 3e-4\ref_p\t\t = %.2f %.2f\n\gen_vel\t\
            t = no \gen_temp\t\t = 320 \gen_seed\t\t = 473529\n\nstraints\t\t = none \
            nstraint_algorithm\t\t = Lincs\nperiodic-molecules = yes\n",VARIABLE_punt->N_sim,
            nstxout,VARIABLE_punt->N_sim,VARIABLE_punt->N_sim,VARIABLE_punt->tau_t,VARIABLE_punt->
            ref_t,Pcoupl,ref_p,ref_p);



    fclose(sim);

}




void Prepara_FicherosParaTermalizacion(struct VARIABLE_struct* VARIABLE_punt,struct CANAL_struct*
     CANAL_punt,struct POL_struct* POL_punt,struct MEMBRANA_struct* MEMBRANA_punt,int pinoffset,int
     ntmpi, int ntomp, int index_simulacion)
{
    float box_x;
    float box_y;
    float box_z;
    char comando[8192];
    FILE *pipe;


    if(fork()==0)
    {
        sprintf(comando,"mkdir ./CarpetaPrincipal_%d\n",index_simulacion);

        pipe=popen(comando,"w");
        pclose(pipe);

        LiberarMemoriaReservada(CANAL_punt,POL_punt,MEMBRANA_punt,VARIABLE_punt); exit(0);
    }
    wait(NULL);

    if(strcmp(VARIABLE_punt->pulling,"si")==0)
    {
        if(strcmp(VARIABLE_punt->Acoplo_presion,"si")==0)
        {
            box_x=2*VARIABLE_punt->N_sim_cte*(1.5+POL_punt->longitud[index_simulacion]);
            box_y=box_x;
            box_z=box_x;
        }

        if(strcmp(VARIABLE_punt->Acoplo_presion,"no")==0)
        {
            box_x=0.47+MEMBRANA_punt->lado;
            box_y=0.47+MEMBRANA_punt->lado;
            box_z=2*VARIABLE_punt->N_sim_cte*(1.5+POL_punt->longitud[index_simulacion]);
        }
    }

    if(strcmp(VARIABLE_punt->pulling,"no")==0) //En este caso la caja es siempre pequenya
    {
        box_x=0.47+MEMBRANA_punt->lado;
        box_y=0.47+MEMBRANA_punt->lado;
        box_z=2*(1.5+POL_punt->longitud[index_simulacion]);
    }

    if(fork()==0)
    {
        sprintf(comando,"gmx editconf -f SISTEMA_conf_%d.gro -o Sistema_%d.gro -box %.2f %.2f %.2f -
             c\n",index_simulacion,index_simulacion,box_x,box_y,box_z);
        pipe=popen(comando,"w");

        pclose(pipe);
```

```
        LiberarMemoriaReservada(CANAL_punt,POL_punt,MEMBRANA_punt,VARIABLE_punt); exit(0);
    }
    wait(NULL);


    EscribeFicheroTopol(MEMBRANA_punt->NumeroDSPC,0,index_simulacion,CANAL_punt->sel,POL_punt->sel,
        MEMBRANA_punt->sel);

    if(fork()==0)
    {
        sprintf(comando,"cp ./Input/minim.mdp ./Input/term.mdp ./Input/Lterm.mdp ./Input/
            dry_martini_v2.1.itp ./Input/martini_v2.2.itp ./Input/water.gro Sistema_%d.gro
            SISTEMA_canal_%d.itp SISTEMA_pol_%d.itp SISTEMA_dspc_%d.itp SISTEMA_index_%d.ndx topol_
            %d.top CarpetaPrincipal_%d\n",index_simulacion,index_simulacion,index_simulacion,
            index_simulacion,index_simulacion,index_simulacion,index_simulacion);

        pipe=popen(comando,"w");
        pclose(pipe);

        LiberarMemoriaReservada(CANAL_punt,POL_punt,MEMBRANA_punt,VARIABLE_punt); exit(0);
    }
    wait(NULL);

    if(strcmp(VARIABLE_punt->conf_externa,"no")==0)
    {
        if(fork()==0)
        {
            sprintf(comando,"cd ./CarpetaPrincipal_%d\ngmx grompp -f minim.mdp -c Sistema_%d.gro -p
                topol_%d.top -o topol_%d.tpr -n SISTEMA_index_%d.ndx -maxwarn 3\n",index_simulacion
                ,index_simulacion,index_simulacion,index_simulacion,index_simulacion);
            pipe=popen(comando,"w");

            pclose(pipe);

            LiberarMemoriaReservada(CANAL_punt,POL_punt,MEMBRANA_punt,VARIABLE_punt); exit(0);
        }
        wait(NULL);


        if(fork()==0)
        {
            sprintf(comando,"cd ./CarpetaPrincipal_%d\ngmx mdrun -s topol_%d.tpr -c Sistema_min_%d.
                gro -o traj_min_%d.trr -rdd 2 -v -ntmpi %d -pin on -pinoffset %d -pinstride 1\n",
                index_simulacion,index_simulacion,index_simulacion,index_simulacion,ntmpi,pinoffset
                );

            pipe=popen(comando,"w");

            pclose(pipe);

            LiberarMemoriaReservada(CANAL_punt,POL_punt,MEMBRANA_punt,VARIABLE_punt); exit(0);
        }
        wait(NULL);
    }
    if(strcmp(VARIABLE_punt->conf_externa,"si")==0)
    {
        if(fork()==0)
        {
            sprintf(comando,"cp ./CarpetaPrincipal_%d/Sistema_%d.gro ./CarpetaPrincipal_%d/
                Sistema_min_%d.gro\n",index_simulacion,index_simulacion,index_simulacion,
                index_simulacion);

            pipe=popen(comando,"w");
            pclose(pipe);

            LiberarMemoriaReservada(CANAL_punt,POL_punt,MEMBRANA_punt,VARIABLE_punt); exit(0);
        }
        wait(NULL);
    }
}


void Simula_Termalizacion(struct CANAL_struct* CANAL_punt,struct POL_struct* POL_punt, struct
    MEMBRANA_struct* MEMBRANA_punt,struct VARIABLE_struct* VARIABLE_punt,int pinoffset,int ntmpi,
    int ntomp,int index_simulacion)
{
    int i;
    float radio_giro_esperado;
    float POL_long_trans;
    float derivada[3]={0,0,0};
    char comando[8192];
    char CarpetaPrincipal_name[256];
    FILE *pipe;

    if(strcmp(VARIABLE_punt->conf_externa,"no")==0)
    {
        sprintf(CarpetaPrincipal_name,"CarpetaPrincipal_%d",index_simulacion);

        if(fork()==0)
        {

            sprintf(comando,"cd ./CarpetaPrincipal_%d\ngmx grompp -f term.mdp -c Sistema_min_%d.gro
                -p topol_%d.top -o topol_%d.tpr -n SISTEMA_index_%d.ndx -maxwarn 3\n",
```

```
            index_simulacion,index_simulacion,index_simulacion,index_simulacion,
            index_simulacion);
        pipe=popen(comando,"w");

        pclose(pipe);

        LiberarMemoriaReservada(CANAL_punt,POL_punt,MEMBRANA_punt,VARIABLE_punt); exit(0);
    }
    wait(NULL);


    if(fork()==0)
    {

        sprintf(comando,"cd ./CarpetaPrincipal_%d\ngmx mdrun -s topol_%d.tpr -c Sistema_rdy_%d.
            gro -o traj_term_%d.trr -rdd 2 -v -ntmpi %d -pin on -pinoffset %d -pinstride 1\n",
            index_simulacion,index_simulacion,index_simulacion,index_simulacion,ntmpi,pinoffset
            );
        pipe=popen(comando,"w");

        pclose(pipe);

        LiberarMemoriaReservada(CANAL_punt,POL_punt,MEMBRANA_punt,VARIABLE_punt); exit(0);
    }
    wait(NULL);

    POL_long_trans=POL_punt->longitud[index_simulacion]-fabs((POL_punt->posicion+1)*0.5*POL_punt
        ->longitud[index_simulacion]+CANAL_punt->bond_reposo*0.5*CANAL_punt->Nb);

    radio_giro_esperado=sqrt(POL_punt->bond_reposo*POL_long_trans*POL_punt->bending_fuerza/(6*
        VARIABLE_punt->ref_t*kB*Navogadro));


    for(i=0;i<1000;i++)
    {
        if(fork()==0)
        {
            sprintf(comando,"cd ./CarpetaPrincipal_%d\ngmx grompp -f term.mdp -c Sistema_rdy_%d.
                gro -p topol_%d.top -o topol_%d.tpr -n SISTEMA_index_%d.ndx -maxwarn 3\n",
                index_simulacion,index_simulacion,index_simulacion,index_simulacion,
                index_simulacion);
            pipe=popen(comando,"w");

            pclose(pipe);

            LiberarMemoriaReservada(CANAL_punt,POL_punt,MEMBRANA_punt,VARIABLE_punt); exit(0);
        }
        wait(NULL);


        if(fork()==0)
        {

            sprintf(comando,"cd ./CarpetaPrincipal_%d\ngmx mdrun -s topol_%d.tpr -c Sistema_rdy_
                %d.gro -o traj_term_%d.trr -rdd 2 -v -ntmpi %d -pin on -pinoffset %d -pinstride
                 1\n",index_simulacion,index_simulacion,index_simulacion,index_simulacion,ntmpi
                ,pinoffset);
            pipe=popen(comando,"w");

            pclose(pipe);

            LiberarMemoriaReservada(CANAL_punt,POL_punt,MEMBRANA_punt,VARIABLE_punt); exit(0);
        }
        wait(NULL);

        VARIABLE_punt->radio_giro=CalculaRadioGiro(index_simulacion,CarpetaPrincipal_name,"
            Sistema_rdy");
        printf("\n\nRadio_giro esperado= %f\nRadio_giro instantaneo= %f\ndiferencia_radio de
            giro = %f\n\n",radio_giro_esperado,VARIABLE_punt->radio_giro,VARIABLE_punt->
            radio_giro-radio_giro_esperado);

        derivada[0]=derivada[1];
        derivada[1]=derivada[2];
        derivada[2]=VARIABLE_punt->radio_giro;


        printf("\nderivada= %f\ttolerancia= %f\n\n",derivada[2]-derivada[0],VARIABLE_punt->
            term_tol_derivada);

        if(VARIABLE_punt->radio_giro-radio_giro_esperado<VARIABLE_punt->term_tolerancia)
        {
            if(fabs((derivada[0]-derivada[2]))<VARIABLE_punt->term_tol_derivada)
                break;
        }
    }
}

if(strcmp(VARIABLE_punt->conf_externa,"si")==0)
{
    if(fork()==0)
    {
        sprintf(comando,"cp ./CarpetaPrincipal_%d/Sistema_min_%d.gro ./CarpetaPrincipal_%d/
            Sistema_rdy_%d.gro\n",index_simulacion,index_simulacion,index_simulacion,
```

```
                    index_simulacion);

                pipe=popen(comando,"w");
                pclose(pipe);

                LiberarMemoriaReservada(CANAL_punt,POL_punt,MEMBRANA_punt,VARIABLE_punt); exit(0);
            }
            wait(NULL);
        }
}

void Prepara_FicherosParaEquilibrado(struct VARIABLE_struct* VARIABLE_punt,struct CANAL_struct*
     CANAL_punt,struct POL_struct* POL_punt,struct MEMBRANA_struct* MEMBRANA_punt,int pinoffset,int
     ntmpi, int ntomp,int index_simulacion)
{
    int i;
    int SISTEMA_Nuevo_n;
    char comando[8192];
    char basura[256];
    char Indices_name[256];
    char Sistema_name[256];
    FILE *SISTEMA_nuevo;
    FILE *append_index;
    FILE *read_Sistema;
    FILE *pipe;

    sprintf(Indices_name,"CarpetaPrincipal_%d/SISTEMA_index_%d.ndx",index_simulacion,
         index_simulacion);
    sprintf(Sistema_name,"CarpetaPrincipal_%d/Sistema_rdy_%d.gro",index_simulacion,index_simulacion)
         ;


    if(VARIABLE_punt->AGUA_sel==1)
    {
        if(fork()==0)
        {
            sprintf(comando,"cd ./CarpetaPrincipal_%d\ngmx solvate -cp Sistema_rdy_%d.gro -cs water.
                 gro -o Sistema_rdy_%d.gro -radius 0.21\n",index_simulacion,index_simulacion,
                 index_simulacion);
            pipe=popen(comando,"w");

            pclose(pipe);

            LiberarMemoriaReservada(CANAL_punt,POL_punt,MEMBRANA_punt,VARIABLE_punt); exit(0);
        }
        wait(NULL);
    }



    append_index=fopen(Indices_name,"a");
    read_Sistema=fopen(Sistema_name,"r");
    fscanf(read_Sistema,"%s\n%d",basura,&SISTEMA_Nuevo_n);
    fclose(read_Sistema);

    fseek(append_index,1,SEEK_END);
    fprintf(append_index,"\n[SISTEMA]\n");
    for(i=0;i<SISTEMA_Nuevo_n;i++)
        fprintf(append_index,"%d\n",i+1);
    fflush(append_index);

    if(VARIABLE_punt->AGUA_sel==1)
    {
        fprintf(append_index,"\n[W]\n");
        for(i=0;i<SISTEMA_Nuevo_n-VARIABLE_punt->SISTEMA_n;i++)
            fprintf(append_index,"%d\n",VARIABLE_punt->SISTEMA_n+i+1);
        fflush(append_index);
    }

    fclose(append_index);

    EscribeFicheroTopol(MEMBRANA_punt->NumeroDSPC,SISTEMA_Nuevo_n-VARIABLE_punt->SISTEMA_n,
         index_simulacion,CANAL_punt->sel,POL_punt->sel,MEMBRANA_punt->sel);



    if(fork()==0)
    {
        sprintf(comando,"cp ./topol_%d.top ./CarpetaPrincipal_%d\n",index_simulacion,
             index_simulacion);

        pipe=popen(comando,"w");
        pclose(pipe);

        LiberarMemoriaReservada(CANAL_punt,POL_punt,MEMBRANA_punt,VARIABLE_punt); exit(0);
    }
    wait(NULL);



    if(fork()==0)
    {
```

```
        sprintf(comando,"cd ./CarpetaPrincipal_%d\ngmx grompp -f minim.mdp -c Sistema_rdy_%d.gro -p
            topol_%d.top -o topol_%d.tpr -n SISTEMA_index_%d.ndx -maxwarn 3\n",index_simulacion,
            index_simulacion,index_simulacion,index_simulacion,index_simulacion);
        pipe=popen(comando,"w");

        pclose(pipe);

        LiberarMemoriaReservada(CANAL_punt,POL_punt,MEMBRANA_punt,VARIABLE_punt); exit(0);
    }
    wait(NULL);


    if(fork()==0)
    {

        sprintf(comando,"cd ./CarpetaPrincipal_%d\ngmx mdrun -s topol_%d.tpr -c Sistema_min_%d.gro -
            o traj_min_%d.trr -rdd 2 -v -ntmpi %d -pin on -pinoffset %d -pinstride 1\n",
            index_simulacion,index_simulacion,index_simulacion,index_simulacion,ntmpi,pinoffset);
        pipe=popen(comando,"w");

        pclose(pipe);

        LiberarMemoriaReservada(CANAL_punt,POL_punt,MEMBRANA_punt,VARIABLE_punt); exit(0);
    }
    wait(NULL);


}

void Simula_Equilibrado(struct CANAL_struct* CANAL_punt,struct POL_struct* POL_punt, struct
    MEMBRANA_struct* MEMBRANA_punt,struct VARIABLE_struct* VARIABLE_punt,int pinoffset,int ntmpi,
    int ntomp, int index_simulacion)
{
    char comando[8192];
    FILE *pipe;


    if(fork()==0)
    {

        sprintf(comando,"cd ./CarpetaPrincipal_%d\ngmx grompp -f Lterm.mdp -c Sistema_min_%d.gro -p
            topol_%d.top -o topol_%d.tpr -n SISTEMA_index_%d.ndx -maxwarn 3\n",index_simulacion,
            index_simulacion,index_simulacion,index_simulacion); //Cambio a la
            carpeta principal para termalizar
        pipe=popen(comando,"w");

        pclose(pipe);

        LiberarMemoriaReservada(CANAL_punt,POL_punt,MEMBRANA_punt,VARIABLE_punt); exit(0);
    }
    wait(NULL);


    if(fork()==0)
    {

        sprintf(comando,"cd ./CarpetaPrincipal_%d\ngmx mdrun -s topol_%d.tpr -c Sistema_term_%d.gro
            -o traj_term_%d.trr -rdd 2 -v -ntmpi %d -pin on -pinoffset %d -pinstride 1\n",
            index_simulacion,index_simulacion,index_simulacion,index_simulacion,ntmpi,pinoffset);
        pipe=popen(comando,"w");


        pclose(pipe);

        LiberarMemoriaReservada(CANAL_punt,POL_punt,MEMBRANA_punt,VARIABLE_punt); exit(0);
    }
    wait(NULL);




    if(fork()==0)
    {
        sprintf(comando,"cp ./CarpetaPrincipal_%d/Sistema_term_%d.gro ./CarpetaPrincipal_%d/
            SISTEMA_index_%d.ndx ./\n",index_simulacion,index_simulacion,index_simulacion,
            index_simulacion);

        pipe=popen(comando,"w");
        pclose(pipe);

        LiberarMemoriaReservada(CANAL_punt,POL_punt,MEMBRANA_punt,VARIABLE_punt); exit(0);
    }
    wait(NULL);

    if(fork()==0)
    {
        sprintf(comando,"rm Sistema_%d.gro SISTEMA_conf_%d.gro\nrm -r CarpetaPrincipal_%d\n",
            index_simulacion,index_simulacion,index_simulacion);

        pipe=popen(comando,"w");
        pclose(pipe);

        LiberarMemoriaReservada(CANAL_punt,POL_punt,MEMBRANA_punt,VARIABLE_punt); exit(0);
```

```
    }
    wait(NULL);

}

void Prepara_FicherosParaTraslocacion(struct CANAL_struct* CANAL_punt,struct POL_struct* POL_punt,
    struct MEMBRANA_struct* MEMBRANA_punt,struct VARIABLE_struct* VARIABLE_punt,int
    index_simulacion, int index_estad)
{
    char comando[8192];
    FILE *pipe;

    if(fork()==0)
    {
        sprintf(comando,"mkdir ./Simulacion__%d_%d\n",index_simulacion,index_estad);

        pipe=popen(comando,"w");
        pclose(pipe);

        LiberarMemoriaReservada(CANAL_punt,POL_punt,MEMBRANA_punt,VARIABLE_punt); exit(0);
    }
    wait(NULL);

    if(fork()==0)
    {
        sprintf(comando,"cp ./Input/martini_md_%d_%d.mdp ./Input/martini_v2.2.itp ./Input/
            dry_martini_v2.1.itp ./SISTEMA_canal_%d.itp ./SISTEMA_pol_%d.itp ./SISTEMA_dspc_%d.itp
            ./SISTEMA_index_%d.ndx ./topol_%d.top ./Sistema_term_%d.gro ./Simulacion__%d_%d\n",
            index_simulacion,index_estad,index_simulacion,index_simulacion,index_simulacion,
            index_simulacion,index_simulacion,index_simulacion,index_simulacion,index_estad);

        pipe=popen(comando,"w");
        pclose(pipe);

        LiberarMemoriaReservada(CANAL_punt,POL_punt,MEMBRANA_punt,VARIABLE_punt); exit(0);
    }

    wait(NULL);
}

void Simula_Traslocacion(struct CANAL_struct* CANAL_punt,struct POL_struct* POL_punt, struct
    MEMBRANA_struct* MEMBRANA_punt,struct VARIABLE_struct* VARIABLE_punt,int pinoffset,int ntmpi,
    int ntomp,int index_simulacion,int index_estad)
{
    int i,j,k;
    float distancia;
    char comando[8192];
    FILE *pipe;

    if(fork()==0)
    {
        sprintf(comando,"cd ./Simulacion__%d_%d\ngmx grompp -f martini_md_%d_%d.mdp -c Sistema_term_
            %d.gro -p topol_%d.top -o topol_%d.tpr -n SISTEMA_index_%d.ndx -maxwarn 3\n",
            index_simulacion,index_estad,index_simulacion,index_estad,index_simulacion,
            index_simulacion,index_simulacion,index_simulacion);
        pipe=popen(comando,"w");

        pclose(pipe);

        LiberarMemoriaReservada(CANAL_punt,POL_punt,MEMBRANA_punt,VARIABLE_punt); exit(0);
    }
    wait(NULL);

    if(fork()==0)
    {

        sprintf(comando,"cd ./Simulacion__%d_%d\ngmx mdrun -s topol_%d.tpr -c Sistema_end_%d_%d.gro
            -o traj_%d_%d.trr -px pullx_%d_%d.xvg -pf pullf_%d_%d.xvg -rdd 2 -v -ntmpi %d -pin on -
            pinoffset %d -pinstride 1\n",index_simulacion,index_estad,index_simulacion,
            index_simulacion,index_estad,index_simulacion,index_estad,index_simulacion,index_estad,
            index_simulacion,index_estad,ntmpi,pinoffset);

        pipe=popen(comando,"w");

        pclose(pipe);

        LiberarMemoriaReservada(CANAL_punt,POL_punt,MEMBRANA_punt,VARIABLE_punt);
        exit(0);
    }
    wait(NULL);


}

void Prepara_FicherosParaAnalisis(struct VARIABLE_struct* VARIABLE_punt,struct CANAL_struct*
    CANAL_punt,struct POL_struct* POL_punt,struct MEMBRANA_struct* MEMBRANA_punt,int
    index_simulacion,int index_estad)
{

    char comando[8192];
    char tiempo_trans_name[256];
    float tiempo_trans;
    FILE *pipe,*tiempo_trans_file;
```

```
    if(fork()==0)
    {


        sprintf(comando,"cd ./Simulacion__%d_%d\ngmx trjconv -f traj_%d_%d.trr -s topol_%d.tpr -o
            traj_analisis_%d_%d.gro -pbc mol -n SISTEMA_index_%d.ndx\n",index_simulacion,
            index_estad,index_simulacion,index_estad,index_simulacion,index_simulacion,index_estad,
            index_simulacion);
        pipe=popen(comando,"w");

        sprintf(comando,"5\n");
        fprintf(pipe,"%s",comando);


        pclose(pipe);

        LiberarMemoriaReservada(CANAL_punt,POL_punt,MEMBRANA_punt,VARIABLE_punt);
        exit(0);
    }

    wait(NULL);


    if(strcmp(VARIABLE_punt->PintaTrayectoria,"si")==0)
    {
        if(fork()==0)
        {

            sprintf(comando,"cd ./Simulacion__%d_%d\ngmx trjconv -f traj_%d_%d.trr -s topol_%d.tpr -
                o traj_view_%d_%d.pdb -pbc mol -n SISTEMA_index_%d.ndx\n",index_simulacion,
                index_estad,index_simulacion,index_estad,index_simulacion,index_simulacion,
                index_estad,index_simulacion);
            pipe=popen(comando,"w");

            sprintf(comando,"4\n");
            fprintf(pipe,"%s",comando);

            pclose(pipe);

            LiberarMemoriaReservada(CANAL_punt,POL_punt,MEMBRANA_punt,VARIABLE_punt); exit(0);
        }
        wait(NULL);
    }


    if(fork()==0)
    {
        sprintf(comando,"cp ./Simulacion__%d_%d/traj_analisis_%d_%d.gro ./Simulacion__%d_%d/pullf_%
            d_%d.xvg ./Simulacion__%d_%d/pullx_%d_%d.xvg ./\n",index_simulacion,index_estad,
            index_simulacion,index_estad,index_simulacion,index_estad,index_simulacion,index_estad,
            index_simulacion,index_estad,index_simulacion,index_estad);

        pipe=popen(comando,"w");
        pclose(pipe);

        LiberarMemoriaReservada(CANAL_punt,POL_punt,MEMBRANA_punt,VARIABLE_punt); exit(0);
    }
    wait(NULL);



    if(strcmp(VARIABLE_punt->PintaTrayectoria,"no")==0)
    {

        if(fork()==0)
        {
            sprintf(comando,"rm -r Simulacion__%d_%d\n",index_simulacion,index_estad);

            pipe=popen(comando,"w");
            pclose(pipe);

            LiberarMemoriaReservada(CANAL_punt,POL_punt,MEMBRANA_punt,VARIABLE_punt); exit(0);
        }
        wait(NULL);

    }

    tiempo_trans=TiempoTraslocacion(CANAL_punt,POL_punt,MEMBRANA_punt,VARIABLE_punt,index_simulacion
        ,index_estad);
    sprintf(tiempo_trans_name,"TiempoTrans_%d_%d.txt",index_simulacion,index_estad);
    tiempo_trans_file=fopen(tiempo_trans_name,"w");
    fprintf(tiempo_trans_file,"%f",tiempo_trans);
    fclose(tiempo_trans_file);
}

void UtilizaEstructuraPredefinida(struct CANAL_struct* CANAL_punt,struct POL_struct* POL_punt,
    struct MEMBRANA_struct* MEMBRANA_punt,struct VARIABLE_struct* VARIABLE_punt, int
    index_simulacion)
{
    char comando[8192];
    FILE* pipe;
```

```
    if(fork()==0)
    {
        sprintf(comando,"rm ./SISTEMA_conf_%d.gro\n",index_simulacion);

        pipe=popen(comando,"w");
        pclose(pipe);

        LiberarMemoriaReservada(CANAL_punt,POL_punt,MEMBRANA_punt,VARIABLE_punt); exit(0);
    }
    wait(NULL);

    if(fork()==0)
    {
        sprintf(comando,"cp ./ConfiguracionesIniciales/%.2f_%.2f__%.1f_%.2f__%.1f_.gro ./
            SISTEMA_conf_%d.gro\n",CANAL_punt->radio,CANAL_punt->bond_reposo,POL_punt->longitud[
            index_simulacion],POL_punt->bond_reposo,MEMBRANA_punt->area,index_simulacion);

        pipe=popen(comando,"w");
        pclose(pipe);

        LiberarMemoriaReservada(CANAL_punt,POL_punt,MEMBRANA_punt,VARIABLE_punt); exit(0);
    }
    wait(NULL);
}


int GarantizoQueTodoEstaBien(struct CANAL_struct* CANAL_punt,struct POL_struct* POL_punt, struct
    MEMBRANA_struct* MEMBRANA_punt,struct VARIABLE_struct* VARIABLE_punt)
{
    int i;
    int inquisidor=0;
    int suma_inquisidor=0;
    char leido[256];
    char modelo_f[256];
    char modelo_x[256];
    char modelo_t[256];
    char file_name[256];
    char comando[256];
    int index_simulacion,index_estad;
    FILE *reader,*pipe;

    sprintf(file_name,"stdout.txt");
    pipe=popen("ls >> stdout.txt\n","w");
    pclose(pipe);

    reader=fopen(file_name,"r");

    if(strcmp(VARIABLE_punt->pulling,"si")==0)
    {
        for(index_simulacion=0;index_simulacion<VARIABLE_punt->N_simulaciones;index_simulacion++)
        {

            for(index_estad=0;index_estad<VARIABLE_punt->N_estadistica;index_estad++)
            {
                sprintf(modelo_f,"pullf_%d_%d.xvg",index_simulacion,index_estad);
                sprintf(modelo_x,"pullx_%d_%d.xvg",index_simulacion,index_estad);
                sprintf(modelo_t,"TiempoTrans_%d_%d.txt",index_simulacion,index_estad);

                inquisidor=0;
                fseek(reader,0,SEEK_SET);
                for(i=0;i<1;i++)
                {
                    fscanf(reader,"%s",leido);
                    if(strcmp(leido,modelo_f)==0)
                        inquisidor++;
                    if(strcmp(leido,modelo_x)==0)
                        inquisidor++;
                    if(strcmp(leido,modelo_t)==0)
                        inquisidor++;

                    if(inquisidor==3)
                        break;

                    if(feof(reader)!=0)
                        break;
                    i=-1;
                }
                if(inquisidor==3)
                    VARIABLE_punt->MatrizSeguridad[index_simulacion][index_estad]=1;

                else
                    VARIABLE_punt->MatrizSeguridad[index_simulacion][index_estad]=0;

                suma_inquisidor+=inquisidor;

                printf("%.2f %%\n",(100.0*(index_estad+index_simulacion*VARIABLE_punt->N_estadistica
                    ))/(VARIABLE_punt->N_simulaciones*VARIABLE_punt->N_estadistica));

            }
            VARIABLE_punt->ArraySeguridad[index_simulacion]=1;
            for(i=0;i<VARIABLE_punt->N_estadistica;i++)
            {
                if(VARIABLE_punt->MatrizSeguridad[index_simulacion][i]==0)
```

```
                {
                    VARIABLE_punt->ArraySeguridad[index_simulacion]=0;
                    break;
                }
            }
        }
    }

    else
    {
        for(index_simulacion=0;index_simulacion<VARIABLE_punt->N_simulaciones;index_simulacion++)
        {
            for(index_estad=0;index_estad<VARIABLE_punt->N_estadistica;index_estad++)
            {
                sprintf(modelo_t,"TiempoTrans_%d_%d.txt",index_simulacion,index_estad);

                inquisidor=0;
                fseek(reader,0,SEEK_SET);
                for(i=0;i<1;i++)
                {
                    fscanf(reader,"%s",leido);

                    if(strcmp(leido,modelo_t)==0)
                        inquisidor++;

                    if(inquisidor==1)
                        break;

                    if(feof(reader)!=0)
                        break;
                    i=-1;
                }
                if(inquisidor==1)
                    VARIABLE_punt->MatrizSeguridad[index_simulacion][index_estad]=1;

                else
                    VARIABLE_punt->MatrizSeguridad[index_simulacion][index_estad]=0;

                suma_inquisidor+=inquisidor;

                printf("%.2f %%\n",(100.0*(index_estad+index_simulacion*VARIABLE_punt->N_estadistica
                    ))/(VARIABLE_punt->N_simulaciones*VARIABLE_punt->N_estadistica));

            }
            VARIABLE_punt->ArraySeguridad[index_simulacion]=1;
            for(i=0;i<VARIABLE_punt->N_estadistica;i++)
            {
                if(VARIABLE_punt->MatrizSeguridad[index_simulacion][i]==0)
                {
                    VARIABLE_punt->ArraySeguridad[index_simulacion]=0;
                    break;
                }
            }
        }
    }



    printf("\n\n");
    for(index_simulacion=0;index_simulacion<VARIABLE_punt->N_simulaciones;index_simulacion++)
    {
        for(index_estad=0;index_estad<VARIABLE_punt->N_estadistica;index_estad++)
            printf("%d ",VARIABLE_punt->MatrizSeguridad[index_simulacion][index_estad]);
        printf("\n");
    }

    fclose(reader);

    sprintf(comando,"rm %s\n",file_name);
    pipe=popen(comando,"w");
    pclose(pipe);


    if(strcmp(VARIABLE_punt->pulling,"si")==0)
    {
        if(suma_inquisidor==3*VARIABLE_punt->N_simulaciones*VARIABLE_punt->N_estadistica)
            return 1;
        else
            return 0;
    }

    else
    {
        if(suma_inquisidor==VARIABLE_punt->N_simulaciones*VARIABLE_punt->N_estadistica)
            return 1;
        else
            return 0;
    }


}
```

(2)

```
#include "MN_Funciones.h"

#define PI 3.1415926536
#define IA 16807
#define IM 2147483647
#define AM (1.0/IM)
#define IQ 127773
#define IR 2836
#define NTAB 32
#define NDIV (1+(IM-1)/NTAB)
#define EPS 1.2e-7
#define RNMX (1.0-EPS)



int main()
{
    int i,j,k,l;
    int index_sim_bond;
    int index_sim_bending;
    int pid_hijo0;
    int pid_padre;
    int pinoffset;
    int ntmpi;
    long idum;

    struct CANAL_struct *CANAL_punt=&CANAL;
    struct VARIABLE_struct *VARIABLE_punt=&VARIABLE;


    printf("\n\nScript: LanzarSimulacion, TFM GDS, Unizar.\n\n");

    idum=-(long)time(NULL);

    Genconf();

    LeerInput(CANAL_punt,VARIABLE_punt);

    GeneraMDP(VARIABLE_punt);


    pid_padre=getpid();
    pid_hijo0=-1;
    ntmpi=1;

    pid_hijo0=fork();
    pinoffset=0;


    if(pid_hijo0==0)
        pid_padre=-1;



    if(pid_padre==-1)
    {
        for(index_sim_bond=0;index_sim_bond<VARIABLE_punt->N_simulaciones;index_sim_bond++)
        {
            for(index_sim_bending=0;index_sim_bending<VARIABLE_punt->N_simulaciones;
                index_sim_bending++)
            {
                GenerarEstructuras(CANAL_punt,VARIABLE_punt,index_sim_bond,index_sim_bending);

                MinimizaEstructura(VARIABLE_punt,CANAL_punt,pinoffset,ntmpi,index_sim_bond,
                    index_sim_bending);

                AnalisisDeModosNormales(VARIABLE_punt,CANAL_punt,pinoffset,ntmpi,index_sim_bond,
                    index_sim_bending);

                ProcesoResultados(CANAL_punt,VARIABLE_punt,index_sim_bond,index_sim_bending);

            }
        }

        LiberarMemoriaReservada(CANAL_punt,VARIABLE_punt); //Aqui libero la memoria retenida por los
            hijos pero NO por el padre!

        return 0; //fin de los hijos: ya estan hechas todas las simulaciones
    }

    wait(NULL);
    wait(NULL);

    AnalisisDeResultados(VARIABLE_punt,CANAL_punt);
    LiberarMemoriaReservada(CANAL_punt,VARIABLE_punt);

    ///FIN DEL PROGRAMA
    return 0;
```

```
}
```

2.1) Librería del programa *Elasticidad*

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>

#define PI 3.1415926536
#define IA 16807
#define IM 2147483647
#define AM (1.0/IM)
#define IQ 127773
#define IR 2836
#define NTAB 32
#define NDIV (1+(IM-1)/NTAB)
#define EPS 1.2e-7
#define RNMX (1.0-EPS)
#define VelocidadLuz_c 3e-2


struct CANAL_struct
{
    double radio;
    double longitudInterior;
    double longitudExterior;
    double *bond_fuerza;
    double bond_reposo;
    double *bending_fuerza;
    double bending_ang;
    int n;
    int M;
    int Nb;
    int NExterior;
    char Borde_poro[256];
    char Cuerpo_poro[256];
} CANAL;


struct VARIABLE_struct
{
    int SISTEMA_n;
    int N_simulaciones;
    int N_estadistica;
    double **ModoNormal;
} VARIABLE;


int Igual(double a, double b);

void SumaVector(double *a, double *b, double *result);

void RestaVector(double *a, double *b, double *result);

double ProductoEscalar(double *a, double *b);

double AnguloEntreVectores(double *a, double *b);

void ProductoVectorial(double *a, double *b, double *result);

void AsignaVector(double *a, double *result);

void Rotacion(double *EjeRotacion, double *Vector, double angulo);

double Distancia(double **Rx,double **Ry, double **Rz,int m1, int n1,int m2, int n2);

void MedVar(double *distribucion, int N, double* media, double* desviacion);

double ran1(long *idum);

void GenerarEstructuras(struct CANAL_struct* CANAL_punt,struct VARIABLE_struct* VARIABLE_punt,int
    index_sim_bond,int index_sim_bending);

void ConstruyeCanal(double CANAL_radio, double CANAL_bond_reposo, double CANAL_bond_fuerza, double
    CANAL_bending_ang, double CANAL_bending_fuerza, int CANAL_Nb,int CANAL_NExterior, int CANAL_M,
    double CANAL_phi, double **CANAL_X, double **CANAL_V, int *CANAL_residuo, int *CANAL_chgrp,
    double *CANAL_box, double *CANAL_Angle, int index_sim_bond,int index_sim_bending,char*
    Borde_poro,char* Cuerpo_poro);

void ConstruyeDisco(double **Rx,double **Ry,double **Rz, double CANAL_bond_reposo,double CANAL_radio
    , double phi, int M, int i);

void MatrizDistancia(double **MD,double **Rx,double **Ry, double **Rz,int M, int Nb);

void MatrizVecinos(int **MV,int M, int Nb,double **MatrizDistancia,double CANAL_bond_reposo);
```

```
void GeneraBond(int **MatrizEnlaces,int **MVecinos, int M, int Nb);

void GeneraAngle(int ***MatrizAngulos,int **MVecinos, int M, int Nb);

void ConstruyeTopol_canal(double CANAL_bond_reposo,double CANAL_bond_fuerza, double
     CANAL_bending_ang, double CANAL_bending_fuerza,int M, int Nb,int NExterior,int **MEnlaces, int
     ***MAngulos, int index_sim_bond,int index_sim_bending, char* Borde_poro, char* Cuerpo_poro);

void MoverEstructura(double **estructura,int N,double theta, double phi, double Rx, double Ry,
     double Rz, long idum, double *Angle);

void Calcula_CM(double **estructura, int N, double *CentroMasa);

void ConstruyeIndices(struct CANAL_struct* CANAL_punt,struct VARIABLE_struct* VARIABLE_punt, int
     index_sim_bond,int index_sim_bending);

void EscribeFicheroTopol(int index_sim_bond,int index_sim_bending);

void Genconf();


void LeerInput(struct CANAL_struct* CANAL_punt, struct VARIABLE_struct* VARIABLE_punt);

void LiberarMemoriaReservada(struct CANAL_struct* CANAL_punt,struct VARIABLE_struct* VARIABLE_punt);


void GeneraMDP(struct VARIABLE_struct* VARIABLE_punt);


void MinimizaEstructura(struct VARIABLE_struct* VARIABLE_punt,struct CANAL_struct* CANAL_punt,int
     pinoffset,int ntmpi, int index_sim_bond,int index_sim_bending);

void Simula_Termalizacion(struct VARIABLE_struct* VARIABLE_punt,int pinoffset,int ntmpi,int
     index_sim_bond,int index_sim_bending);

void AnalisisDeModosNormales(struct VARIABLE_struct* VARIABLE_punt,struct CANAL_struct* CANAL_punt,
     int pinoffset,int ntmpi,int index_sim_bond,int index_sim_bending);

void Prepara_FicherosParaAnalisis(struct VARIABLE_struct* VARIABLE_punt,struct CANAL_struct*
     CANAL_punt,char* CarpetaSimulacion,int index_sim_bond,int index_sim_bending);

void ProcesoResultados(struct CANAL_struct* CANAL_punt,struct VARIABLE_struct* VARIABLE_punt,int
     index_sim_bond,int index_sim_bending);

void AnalisisDeResultados(struct VARIABLE_struct* VARIABLE_punt, struct CANAL_struct* CANAL_punt);

int AnalisisDeAutovectores(struct VARIABLE_struct* VARIABLE_punt, struct CANAL_struct* CANAL_punt,
     int index_sim_bond,int index_sim_bending);

void OrientarAutovector(struct VARIABLE_struct* VARIABLE_punt, struct CANAL_struct* CANAL_punt,
     double *referencia,double* Autovector);



/*************************
 *                       *
 * FUNCIONES GENERALES   *
 *                       *
 *************************/
///FUNCIONES OPERATIVAS/////////////////////////////////////////////////////////////////
int Igual(double a, double b){   //Devuelve 1 si son iguales, 0 si son diferentes
    double epsilon=0.00000001;

    if (fabs(a-b)<epsilon)
        return 1;

    else
        return 0;
}


void SumaVector(double *a, double *b, double *result){
    int i;
    for(i=0;i<3;i++)
        result[i]=a[i]+b[i];
}

void RestaVector(double *a, double *b, double *result){
    int i;
    for(i=0;i<3;i++)
        result[i]=a[i]-b[i];
}

double ProductoEscalar(double *a, double *b){
    double result=0;
    int i;
    for(i=0;i<3;i++){
        result+=a[i]*b[i];
    }

    return result;
}
```

```
double AnguloEntreVectores(double *a, double *b){
    int i;
    double mod_a,mod_b,angulo;
    mod_a=sqrt(ProductoEscalar(a,a));
    mod_b=sqrt(ProductoEscalar(b,b));

    angulo=acos(ProductoEscalar(a,b)/(mod_a*mod_b));
    return angulo;
}

void ProductoVectorial(double *a, double *b, double *result){

    result[0]=a[1]*b[2]-a[2]*b[1];
    result[1]=a[2]*b[0]-a[0]*b[2];
    result[2]=a[0]*b[1]-a[1]*b[0];

}

void AsignaVector(double *a, double *result){
    result[0]=a[0];
    result[1]=a[1];
    result[2]=a[2];
}

void Rotacion(double *EjeRotacion, double *Vector, double angulo){
    int i,j;
    double MatrizRotacion[3][3];
    double Vector_auxiliar[3]={0,0,0};
    double mod_Eje;

    angulo=angulo*PI/180; ///Convertir angulos a radianes (para operar con math.h)
    mod_Eje=sqrt(ProductoEscalar(EjeRotacion, EjeRotacion)); ///Modulo del eje de rotacion. Debe ser
        1, pero por si acaso lo normalizo antes de seguir.

    EjeRotacion[0]=EjeRotacion[0]/mod_Eje;
    EjeRotacion[1]=EjeRotacion[1]/mod_Eje;
    EjeRotacion[2]=EjeRotacion[2]/mod_Eje;

    MatrizRotacion[0][0]=cos(angulo)+EjeRotacion[0]*EjeRotacion[0]*(1-cos(angulo));
    MatrizRotacion[0][1]=EjeRotacion[0]*EjeRotacion[1]*(1-cos(angulo))-EjeRotacion[2]*sin(angulo);
    MatrizRotacion[0][2]=EjeRotacion[0]*EjeRotacion[2]*(1-cos(angulo))+EjeRotacion[1]*sin(angulo);
    MatrizRotacion[1][0]=EjeRotacion[0]*EjeRotacion[1]*(1-cos(angulo))+EjeRotacion[2]*sin(angulo);
    MatrizRotacion[1][1]=cos(angulo)+EjeRotacion[1]*EjeRotacion[1]*(1-cos(angulo));
    MatrizRotacion[1][2]=EjeRotacion[1]*EjeRotacion[2]*(1-cos(angulo))-EjeRotacion[0]*sin(angulo);
    MatrizRotacion[2][0]=EjeRotacion[0]*EjeRotacion[2]*(1-cos(angulo))-EjeRotacion[1]*sin(angulo);
    MatrizRotacion[2][1]=EjeRotacion[1]*EjeRotacion[2]*(1-cos(angulo))+EjeRotacion[0]*sin(angulo);
    MatrizRotacion[2][2]=cos(angulo)+EjeRotacion[2]*EjeRotacion[2]*(1-cos(angulo)); ///Matriz de
        rotacion construida.

    for(i=0;i<3;i++){
        for(j=0;j<3;j++)
            Vector_auxiliar[i]+=MatrizRotacion[i][j]*Vector[j];
    }

    Vector[0]=Vector_auxiliar[0];
    Vector[1]=Vector_auxiliar[1];
    Vector[2]=Vector_auxiliar[2];

}

double Distancia(double **Rx,double **Ry, double **Rz,int m1, int n1,int m2, int n2){
    return(sqrt((Rx[m1][n1]-Rx[m2][n2])*(Rx[m1][n1]-Rx[m2][n2])+(Ry[m1][n1]-Ry[m2][n2])*(Ry[m1][n1]-
        Ry[m2][n2])+(Rz[m1][n1]-Rz[m2][n2])*(Rz[m1][n1]-Rz[m2][n2])));
}

void MedVar(double *distribucion, int N, double* media, double* desviacion)
{
    int i;
    double promedio=0;
    double varianza=0;


    for(i=0;i<N;i++)
        promedio+=distribucion[i];

    promedio=promedio/N;

    for(i=0;i<N;i++)
    {
        varianza+=(distribucion[i]-promedio)*(distribucion[i]-promedio);
    }
    varianza=varianza/(N-1);

    *media=promedio;
    *desviacion=sqrt(varianza);
}

double ran1(long *idum){
    int j;
    long k;
    static long iy=0;
    static long iv[NTAB];
    double temp;
```

```
    if (*idum <= 0 || !iy) {
        if (-(*idum) < 1) *idum=1;
        else *idum = -(*idum);

        for (j=NTAB+7;j>=0;j--) {
            k=(*idum)/IQ;
            *idum=IA*(*idum-k*IQ)-IR*k;
            if (*idum < 0)
                *idum += IM;
            if (j < NTAB)
                iv[j] = *idum;
        }
        iy=iv[0];
    }
    k=(*idum)/IQ;
    *idum=IA*(*idum-k*IQ)-IR*k;
    if (*idum < 0) *idum += IM;
    j=iy/NDIV;
    iy=iv[j];
    iv[j] = *idum;
    if ((temp=AM*iy) > RNMX) return RNMX;
    else return temp;
}


///CONSTRUCCION DE ESTRUCTURAS/////////////////////////////////////
void GenerarEstructuras(struct CANAL_struct* CANAL_punt,struct VARIABLE_struct* VARIABLE_punt,int
    index_sim_bond,int index_sim_bending)
{
    int i,j,k;
    char comando[512];

    double CANAL_phi;
    char CANAL_name[256];
    int *CANAL_residuo;
    int *CANAL_chgrp;
    int *CANAL_index;
    double **CANAL_X;
    double **CANAL_V;
    double CANAL_box[3];
    double CANAL_Angle[2];

    int AtomNumber=0;
    char Configuracion_name[256];
    long idum=-(long)time(NULL);
    FILE* SISTEMA_nuevo;
    FILE *pipe;



    sprintf(Configuracion_name,"SISTEMA_conf_%d_%d.gro",index_sim_bond,index_sim_bending);
    SISTEMA_nuevo=fopen(Configuracion_name,"w");


    ///------------------CANAL-----------------------------------------------------
    CANAL_punt->Nb=(int)((CANAL_punt->longitudInterior+2*CANAL_punt->longitudExterior)/CANAL_punt->
        bond_reposo);
    CANAL_punt->NExterior=(int)(CANAL_punt->longitudExterior/CANAL_punt->bond_reposo);
    CANAL_phi=acos(1-CANAL_punt->bond_reposo*CANAL_punt->bond_reposo/(2*CANAL_punt->radio*CANAL_punt
        ->radio));
    CANAL_punt->M=(int)(2*PI/CANAL_phi);
    CANAL_punt->n=CANAL_punt->Nb*CANAL_punt->M;




    CANAL_residuo=(int*)malloc(sizeof(int)*CANAL_punt->n);


    CANAL_chgrp=(int*)malloc(sizeof(int)*CANAL_punt->n);
    CANAL_index=(int*)malloc(sizeof(int)*CANAL_punt->n);

    CANAL_X=(double**)malloc(sizeof(double*)*CANAL_punt->n);
    for(i=0;i<CANAL_punt->n;i++)
        *(CANAL_X+i)=(double*)malloc(sizeof(double)*3);

    CANAL_V=(double**)malloc(sizeof(double*)*CANAL_punt->n);
    for(i=0;i<CANAL_punt->n;i++)
        *(CANAL_V+i)=(double*)malloc(sizeof(double)*3);


    printf("\n\nPORO\n------------------\n");

    ConstruyeCanal(CANAL_punt->radio,CANAL_punt->bond_reposo,CANAL_punt->bond_fuerza[index_sim_bond
        ],CANAL_punt->bending_ang,CANAL_punt->bending_fuerza[index_sim_bending],CANAL_punt->Nb,
        CANAL_punt->NExterior,CANAL_punt->M,CANAL_phi,CANAL_X,CANAL_V,CANAL_residuo,CANAL_chgrp,
        CANAL_box,CANAL_Angle,index_sim_bond,index_sim_bending,CANAL_punt->Borde_poro,CANAL_punt->
        Cuerpo_poro);

    MoverEstructura(CANAL_X,CANAL_punt->n,0, 0, -0.4, -0.4, 0, idum,CANAL_Angle);
```

```
    for(i=0;i<CANAL_punt->n;i++){
        AtomNumber++;
        CANAL_index[i]=AtomNumber;
    }



    ///ESCRITURA DEL SISTEMA:
    VARIABLE_punt->SISTEMA_n=CANAL_punt->n;

    fprintf(SISTEMA_nuevo,"%s\n","Sistema");

    fprintf(SISTEMA_nuevo,"%d\n",VARIABLE_punt->SISTEMA_n);

    for(i=0;i<CANAL_punt->n;i++){
        if(i/CANAL_punt->M<CANAL_punt->NExterior || i/CANAL_punt->M+1>CANAL_punt->Nb-CANAL_punt->
            NExterior)
            fprintf(SISTEMA_nuevo,"%5d %-5s %5c %5d %8.3f %8.3f %8.3f %8.3f %8.3f %8.3f\n",CANAL_residuo[i],"
                CHNNL",'G',CANAL_index[i],CANAL_X[i][0],CANAL_X[i][1],CANAL_X[i][2],CANAL_V[i][0],
                CANAL_V[i][1],CANAL_V[i][2]);
        else
            fprintf(SISTEMA_nuevo,"%5d %-5s %5c %5d %8.3f %8.3f %8.3f %8.3f %8.3f %8.3f\n",CANAL_residuo[i],"
                CHNNL",'C',CANAL_index[i],CANAL_X[i][0],CANAL_X[i][1],CANAL_X[i][2],CANAL_V[i][0],
                CANAL_V[i][1],CANAL_V[i][2]);
    }



    fprintf(SISTEMA_nuevo,"%lf\t%lf\t%lf\n",CANAL_box[0],CANAL_box[1],CANAL_box[2]);



    ///ESCRITURA DE INDICES
    ConstruyeIndices(CANAL_punt,VARIABLE_punt,index_sim_bond,index_sim_bending);



    ///LIBERACION DE MEMORIA
    free(CANAL_residuo);

    free(CANAL_chgrp);
    free(CANAL_index);

    for(i=0;i<CANAL_punt->n;i++)
        free(*(CANAL_X+i));
    free(CANAL_X);

    for(i=0;i<CANAL_punt->n;i++)
        free(*(CANAL_V+i));
    free(CANAL_V);


    fflush(SISTEMA_nuevo);
    fclose(SISTEMA_nuevo);

}


///Construccion del Canal

void ConstruyeCanal(double CANAL_radio, double CANAL_bond_reposo, double CANAL_bond_fuerza, double
    CANAL_bending_ang, double CANAL_bending_fuerza, int CANAL_Nb,int CANAL_NExterior, int CANAL_M,
    double CANAL_phi, double **CANAL_X, double **CANAL_V, int *CANAL_residuo, int *CANAL_chgrp,
    double *CANAL_box, double *CANAL_Angle, int index_sim_bond,int index_sim_bending,char*
    Borde_poro,char* Cuerpo_poro){

    ///Definicion de variables auxiliares (para la programacion)
    int n,m,i,j;

    ///Definicion del contenedor de posiciones del sistema
    double **Rx;
    double **Ry;
    double **Rz;
    double **MDistancia;    //Matriz de distancias.
    int **MVecinos;
    int **MEnlaces;
    int ***MAngulos;


    ///Definicion de parametros del canal*****************
    printf("\nNOTA: se han introducido los parametros:\nCANAL_bond_reposo=%.2f\nCANAL_radio=%.1f\
        nLongitud=%d\n\n",CANAL_bond_reposo,CANAL_radio,CANAL_Nb);
    printf("Estos parametros suponen:\nAmplitud_angular=%.2f\tBeads/vuelta=%d\n\n",CANAL_phi,CANAL_M
        );
    CANAL_phi=2*PI/CANAL_M;
    CANAL_radio=CANAL_bond_reposo*sqrt(1/(2*(1-cos(CANAL_phi))));
    printf("Para ajustar el cilindro, se modifican los parametros:\nAmplitud angular'=%.2f\
        tCANAL_radio'=%.3f\n\n",CANAL_phi,CANAL_radio);



    ///Asigancion de memoria al sistema*****************
```

```
Rx=(double**)malloc(sizeof(double*)*CANAL_M);
Ry=(double**)malloc(sizeof(double*)*CANAL_M);
Rz=(double**)malloc(sizeof(double*)*CANAL_M);
for(i=0;i<CANAL_M;i++){
    *(Rx+i)=(double*)malloc(sizeof(double)*CANAL_Nb);
    *(Ry+i)=(double*)malloc(sizeof(double)*CANAL_Nb);
    *(Rz+i)=(double*)malloc(sizeof(double)*CANAL_Nb);
}

MDistancia=(double**)malloc(sizeof(double*)*CANAL_M*CANAL_Nb);
for(i=0;i<(CANAL_M*CANAL_Nb);i++)
    *(MDistancia+i)=(double*)malloc(sizeof(double)*CANAL_M*CANAL_Nb);

MVecinos=(int**)malloc(sizeof(int*)*CANAL_M*CANAL_Nb);
for(i=0;i<CANAL_M*CANAL_Nb;i++)
    *(MVecinos+i)=(int*)malloc(sizeof(int)*CANAL_M*CANAL_Nb);

MEnlaces=(int**)malloc(sizeof(int*)*CANAL_M*CANAL_Nb);
for(i=0;i<(CANAL_M*CANAL_Nb);i++)
    *(MEnlaces+i)=(int*)malloc(sizeof(int)*CANAL_M*CANAL_Nb);


MAngulos=(int***)malloc(sizeof(int**)*CANAL_M*CANAL_Nb);
for(i=0;i<(CANAL_M*CANAL_Nb);i++)
    *(MAngulos+i)=(int**)malloc(sizeof(int*)*CANAL_M*CANAL_Nb);

for(i=0;i<CANAL_M*CANAL_Nb;i++){
    for(j=0;j<CANAL_M*CANAL_Nb;j++)
        *(*(MAngulos+i)+j)=(int*)malloc(sizeof(int)*CANAL_M*CANAL_Nb);
}



///Construccion del sistema
for(i=0;i<CANAL_Nb;i++)
    ConstruyeDisco(Rx,Ry,Rz,CANAL_bond_reposo,CANAL_radio,CANAL_phi,CANAL_M,i); //Con este bucle
        rellenamos las matrices R

MatrizDistancia(MDistancia,Rx,Ry,Rz,CANAL_M,CANAL_Nb);

MatrizVecinos(MVecinos,CANAL_M,CANAL_Nb,MDistancia,CANAL_bond_reposo);

GeneraBond(MEnlaces,MVecinos,CANAL_M,CANAL_Nb);
GeneraAngle(MAngulos,MVecinos,CANAL_M,CANAL_Nb);
ConstruyeTopol_canal(CANAL_bond_reposo,CANAL_bond_fuerza,CANAL_bending_ang,CANAL_bending_fuerza,
    CANAL_M,CANAL_Nb,CANAL_NExterior,MEnlaces,MAngulos,index_sim_bond,index_sim_bending,
    Borde_poro,Cuerpo_poro);


for(n=0;n<CANAL_Nb;n++){
    for(m=0;m<CANAL_M;m++){
        CANAL_X[n*CANAL_M+m][0]=Rx[m][n];
        CANAL_X[n*CANAL_M+m][1]=Ry[m][n];
        CANAL_X[n*CANAL_M+m][2]=Rz[m][n];

        CANAL_V[n*CANAL_M+m][0]=0.0;
        CANAL_V[n*CANAL_M+m][1]=0.0;
        CANAL_V[n*CANAL_M+m][2]=0.0;

        CANAL_residuo[n*CANAL_M+m]=n+1;

        CANAL_chgrp[n*CANAL_M+m]=n*CANAL_M+m+1;
    }
}

CANAL_box[0]=fabs(Rx[CANAL_M-1][CANAL_Nb-1]*1.1);
CANAL_box[1]=fabs(Ry[CANAL_M-1][CANAL_Nb-1]*1.1);
CANAL_box[2]=fabs(Rz[CANAL_M-1][CANAL_Nb-1]*1.1);

CANAL_Angle[0]=0.5*PI;
CANAL_Angle[1]=0;


///Liberar memoria
for(i=0;i<CANAL_M;i++){
    free(*(Rx+i));
    free(*(Ry+i));
    free(*(Rz+i));
}

free(Rx);
free(Ry);
free(Rz);


for(i=0;i<(CANAL_M*CANAL_Nb);i++)
    free(*(MDistancia+i));

free(MDistancia);

for(i=0;i<CANAL_M*CANAL_Nb;i++)
    free(*(MVecinos+i));
free(MVecinos);
```

```
    for(i=0;i<CANAL_M*CANAL_Nb;i++)
        free(*(MEnlaces+i));
    free(MEnlaces);


    for(i=0;i<CANAL_M*CANAL_Nb;i++)
        for(j=0;j<CANAL_M*CANAL_Nb;j++)
            free(*(*(MAngulos+j)+i));

    for(i=0;i<CANAL_M*CANAL_Nb;i++)
        free(*(MAngulos+i));

    free(MAngulos);

}

void ConstruyeDisco(double **Rx,double **Ry,double **Rz, double CANAL_bond_reposo,double CANAL_radio
    , double phi, int M, int i){
    int m;
    double phi_inicial=0;

    if(i%2==0)
        phi_inicial=0;
    else
        phi_inicial=0.5*phi;


    for(m=0;m<M;m++){
        Rx[m][i]=CANAL_bond_reposo*i;
        Ry[m][i]=CANAL_radio*cos(m*phi+phi_inicial);
        Rz[m][i]=CANAL_radio*sin(m*phi+phi_inicial);

    }

}


void MatrizDistancia(double **MD,double **Rx,double **Ry, double **Rz,int M, int Nb){
    int i,j,m1,n1,m2,n2;

    for(i=0;i<(M*Nb);i++){
        for(j=0;j<(M*Nb);j++){
            m1=i%M;
            n1=i/M;
            m2=j%M;
            n2=j/M; //Estas cuatro lineas convierten el indice en los valores m,n

            MD[i][j]=Distancia(Rx,Ry,Rz,m1,n1,m2,n2);
        }
    }
}


void MatrizVecinos(int **MV,int M, int Nb,double **MatrizDistancia,double CANAL_bond_reposo){

    int i,j,m1,n1,m2,n2;
    float cota_dist=sqrt(2.0)*CANAL_bond_reposo;

    for(i=0;i<M*Nb;i++){
        for(j=0;j<M*Nb;j++)
            MV[i][j]=0;
    }

    for(i=0;i<M*Nb;i++){
        m1=i%M;
        n1=i/M;
        for(j=0;j<M*Nb;j++){
            m2=j%M;
            n2=j/M;
            if(i!=j && MatrizDistancia[i][j]<cota_dist)
                MV[i][j]=1;


            if(i!=j && n1==n2 && m1==0 && m2==M-2)
                MV[i][j]=2;

            if(i!=j && n1==n2 && m1==1 && m2==M-1)
                MV[i][j]=2;

            if(i!=j && n1==n2 && m1==m2+2)
                MV[i][j]=2;

            if(i!=j && m1==m2-1 && n1==n2-2)
                MV[i][j]=2;

            if(i!=j && m1==m2+1 && n1==n2-2)
                MV[i][j]=2;

            if(i!=j && m1==M-1 && m2==0 && n1==n2-2)
                MV[i][j]=2;

            if(i!=j && m1==0 && m2==M-1 && n1==n2-2)
```

```
                MV[i][j]=2;


            }
        }

}

void GeneraBond(int **MatrizEnlaces,int **MVecinos, int M, int Nb){
    int i,j;

    for(i=0;i<M*Nb;i++){
        for(j=0;j<M*Nb;j++)
            MatrizEnlaces[i][j]=0;
    }

    for(i=0;i<M*Nb;i++){
        for(j=0;j<M*Nb;j++){
            if(MatrizEnlaces[i][j]==0 && MVecinos[i][j]==1){
                MatrizEnlaces[i][j]=1;
                MatrizEnlaces[j][i]=-1; //De este modo marco el enlace inverso para no repetirlo
                    despues!
            }
        }
    }

}

void GeneraAngle(int ***MatrizAngulos,int **MVecinos, int M, int Nb){
    int i,j,k;

    for(i=0;i<M*Nb;i++){
        for(j=0;j<M*Nb;j++){
            for(k=0;k<M*Nb;k++)
                MatrizAngulos[i][j][k]=0;
        }
    }

    for(i=0;i<M*Nb;i++){
        for(j=0;j<M*Nb;j++){
            for(k=0;k<M*Nb;k++){
                if(MatrizAngulos[i][j][k]==0 && MVecinos[i][j]==1 && MVecinos[j][k]==1 && MVecinos[i
                    ][k]==2){
                    MatrizAngulos[i][j][k]=1;
                    MatrizAngulos[i][k][j]=-1;
                    MatrizAngulos[j][i][k]=-1;
                    MatrizAngulos[k][i][j]=-1;
                    MatrizAngulos[j][k][i]=-1;
                    MatrizAngulos[k][j][i]=-1; //De este modo marco el enlace inverso para no
                        repetirlo despues!
                }
            }
        }
    }

}

void ConstruyeTopol_canal(double CANAL_bond_reposo,double CANAL_bond_fuerza, double
    CANAL_bending_ang, double CANAL_bending_fuerza,int M, int Nb,int NExterior,int **MEnlaces, int
    ***MAngulos, int index_sim_bond,int index_sim_bending, char* Borde_poro, char* Cuerpo_poro){

    int i,j,k;
    FILE *topol;
    char CanalTopol_name[256];

    sprintf(CanalTopol_name,"SISTEMA_canal_%d_%d.itp",index_sim_bond,index_sim_bending);
    topol=fopen(CanalTopol_name,"w");

    fprintf(topol,";Canal proteico para simulacion de traslocacion de DNA. TFM Guillermo Diez.\n\n")
        ;
    fprintf(topol,"[moleculetype]\n;name\texclusion\nCANAL\t1\n\n");

    fprintf(topol,"[atoms]\n;nr\tattype\tresnr\tresidue\tatom\tcgnr\tcharge\tmass\n");
    for(i=0;i<Nb;i++){
        for(j=0;j<M;j++){
            if(i<NExterior || i+1>Nb-NExterior)
                fprintf(topol,"%d\t%s\t%d\tCHNNL\tQ0\t%d\t%.3f\t;\n",j+M*i+1,Borde_poro,i+1,j+M*i
                    +1,0.0); //OJO! Donde pone "Q0" normalmente pone "P5"!!
            else
                fprintf(topol,"%d\t%s\t%d\tCHNNL\tC1\t%d\t%.3f\t;\n",j+M*i+1,Cuerpo_poro,i+1,j+M*i
                    +1,0.0);
        }
    }


    fprintf(topol,"\n\n");
    fprintf(topol,"[bonds]\n;i\tj\tfunc\tb_0\tk_b\n");
    for(i=0;i<M*Nb;i++){
        for(j=0;j<M*Nb;j++){
            if(MEnlaces[i][j]==1)
                fprintf(topol,"%d\t%d\t%d\t%.3f\t%.1f\n",i+1,j+1,1,CANAL_bond_reposo,
                    CANAL_bond_fuerza);
        }
```

```
    }

    fprintf(topol,"\n\n");
    fprintf(topol,"[angles]\n;i\tj\tk\tfunc\ttheta_0\tk_theta\n");
    for(i=0;i<M*Nb;i++){
        for(j=0;j<M*Nb;j++){
            for(k=0;k<M*Nb;k++){
                if(MAngulos[i][j][k]==1)
                    fprintf(topol,"%d\t%d\t%d\t%d\t%.3f\t%.1f\n",i+1,j+1,k+1,1,CANAL_bending_ang,
                        CANAL_bending_fuerza);
            }
        }
    }

    fclose(topol);
}


///Desplazamiento de estructuras
void MoverEstructura(double **estructura,int N,double theta, double phi, double Rx, double Ry,
    double Rz, long idum, double *Angle){ ///Mueve una estructura a la orientacion (theta,phi),
    posicion R
    int i;
    double CentroMasa[3]={0,0,0};
    double delta_theta, delta_phi;
    double Eje_rot_phi[3]={0,0,1};
    double Eje_rot_theta[3];
    double NuevaDireccion[3]={sin(theta)*cos(phi),sin(theta)*sin(phi),cos(theta)};
    double VectorDirector[3];
    double Mod=0;

    Calcula_CM(estructura,N,CentroMasa);

    VectorDirector[0]=sin(Angle[0])*cos(Angle[1]);
    VectorDirector[1]=sin(Angle[0])*sin(Angle[1]);
    VectorDirector[2]=cos(Angle[0]);


    delta_theta=theta-180*Angle[0]/PI;
    delta_phi=phi-180*Angle[1]/PI;


    Rotacion(Eje_rot_phi,VectorDirector,delta_phi);
    for(i=0;i<N;i++)
        Rotacion(Eje_rot_phi,*(estructura+i),delta_phi);

    ProductoVectorial(VectorDirector,NuevaDireccion,Eje_rot_theta);
    Mod=sqrt(ProductoEscalar(Eje_rot_theta,Eje_rot_theta));
    if(Mod<0.00001 && Mod >-0.00001){
        Mod=1.0;
        Eje_rot_theta[0]=NuevaDireccion[0];
        Eje_rot_theta[1]=NuevaDireccion[1];
        Eje_rot_theta[2]=NuevaDireccion[2];
    }

    Eje_rot_theta[0]=Eje_rot_theta[0]/Mod;
    Eje_rot_theta[1]=Eje_rot_theta[1]/Mod;
    Eje_rot_theta[2]=Eje_rot_theta[2]/Mod;


    Rotacion(Eje_rot_theta,VectorDirector,delta_theta);
    for(i=0;i<N;i++)
        Rotacion(Eje_rot_theta,*(estructura+i),delta_theta);

    for(i=0;i<N;i++){
        estructura[i][0]=estructura[i][0]+Rx;
        estructura[i][1]=estructura[i][1]+Ry;
        estructura[i][2]=estructura[i][2]+Rz;
    }


    Angle[0]=PI/180*theta;
    Angle[1]=PI/180*phi;

} ///Coloca el CM de "estructura" en Rx,Ry,Rz; y la orienta en thtea, phi.

void Calcula_CM(double **estructura, int N, double *CentroMasa){
    int i;

    CentroMasa[0]=0;
    CentroMasa[1]=0;
    CentroMasa[2]=0;


    for(i=0;i<N;i++){
        CentroMasa[0]+=estructura[i][0];
        CentroMasa[1]+=estructura[i][1];
        CentroMasa[2]+=estructura[i][2];
    }

    CentroMasa[0]=CentroMasa[0]/((double)N);
    CentroMasa[1]=CentroMasa[1]/((double)N);
    CentroMasa[2]=CentroMasa[2]/((double)N); ///Aqui esta calculado el centro de masas.
```

```
    for(i=0;i<N;i++){
        estructura[i][0]=estructura[i][0]-CentroMasa[0];
        estructura[i][1]=estructura[i][1]-CentroMasa[1];
        estructura[i][2]=estructura[i][2]-CentroMasa[2];
    }
}


///Construccion de indices y topologia
void ConstruyeIndices(struct CANAL_struct* CANAL_punt,struct VARIABLE_struct* VARIABLE_punt, int
    index_sim_bond,int index_sim_bending){
    int i;
    FILE *punte;
    char Indices_name[256];

    sprintf(Indices_name,"SISTEMA_index_%d_%d.ndx",index_sim_bond,index_sim_bending);
    punte=fopen(Indices_name,"w");


    fprintf(punte, "[CHNNL]\n");
    for(i=0;i<CANAL_punt->n;i++)
        fprintf(punte,"%d\n",i+1);


    fprintf(punte, "\n[SECO]\n");
    for(i=0;i<VARIABLE_punt->SISTEMA_n;i++)
        fprintf(punte,"%d\n",i+1);


    fclose(punte);
}

void EscribeFicheroTopol(int index_sim_bond,int index_sim_bending)
{
    FILE *topol;
    char Topol_name[256];

    sprintf(Topol_name,"topol_%d_%d.top",index_sim_bond,index_sim_bending);

    topol=fopen(Topol_name,"w");


    fprintf(topol,"#include %cdry_martini_v2.1.itp%c\n",34,34);

    fprintf(topol,"#include %cSISTEMA_canal_%d_%d.itp%c\n",34,index_sim_bond,index_sim_bending,34);

    fprintf(topol,"[system]\nCanalProteico\n\n[molecules]\n");

    fprintf(topol,"CANAL\t1\n");

    fclose(topol);

}


///CONFIGURACION DEL EXPERIMENTO//////////////////////////////
void Genconf()
{
    int i;
    FILE* genconf;
    FILE* Experimento;
    FILE* pipe;

    int N_simulaciones;
    int N_term;

    char Borde_poro[256];
    char Cuerpo_poro[256];
    char Acoplo_presion[256];
    char basura[256];

    double CANAL_bond_fuerza_ini;
    double CANAL_bond_fuerza_fin;
    double d_CANAL_bond_fuerza;

    double CANAL_bond_reposo;
    double CANAL_bending_ang;

    double CANAL_bending_fuerza_ini;
    double CANAL_bending_fuerza_fin;
    double d_CANAL_bending_fuerza;

    double CANAL_radio;
    double temperatura,tau_t;

    char flag;


    pipe=popen("bash","w");
    fprintf(pipe,"mkdir -p Input\n");
    fflush(pipe);
    pclose(pipe);
```

```
    Experimento=fopen("ModosNormales.txt","r");

    fscanf(Experimento,"%s %s %d",basura,basura,&N_simulaciones);

    fscanf(Experimento,"%s %s %lf",basura,basura,&CANAL_radio);

    fscanf(Experimento,"%s %s %s",basura,basura,Borde_poro);

    fscanf(Experimento,"%s %s %s",basura,basura,Cuerpo_poro);

    fscanf(Experimento,"%s %s %lf",basura,basura,&CANAL_bond_reposo);

    fscanf(Experimento,"%s %s %lf",basura,basura,&CANAL_bending_ang);

    fscanf(Experimento,"%s %s %lf",basura,basura,&CANAL_bond_fuerza_ini);

    fscanf(Experimento,"%s %s %lf",basura,basura,&CANAL_bond_fuerza_fin);

    fscanf(Experimento,"%s %s %lf",basura,basura,&CANAL_bending_fuerza_ini);

    fscanf(Experimento,"%s %s %lf",basura,basura,&CANAL_bending_fuerza_fin);



    fclose(Experimento);


    genconf=fopen("./Input/genconf.txt","w");


    if(N_simulaciones>1)
    {
        d_CANAL_bond_fuerza=exp(log(CANAL_bond_fuerza_fin/CANAL_bond_fuerza_ini)/(N_simulaciones-1))
            ;
        d_CANAL_bending_fuerza=exp(log(CANAL_bending_fuerza_fin/CANAL_bending_fuerza_ini)/(
            N_simulaciones-1));
    }
    else
    {
        d_CANAL_bond_fuerza=1;
        d_CANAL_bending_fuerza=1;
    }


    fprintf(genconf,"N_simulaciones\t=\t%d\n",N_simulaciones);

    fprintf(genconf,"BordePoro\t=\t%s\n",Borde_poro);
    fprintf(genconf,"CuerpoPoro\t=\t%s\n",Cuerpo_poro);


    fprintf(genconf,"CANAL_radio\t=\t%.2f\n",CANAL_radio);

    fprintf(genconf,"CANAL_longitud_Interior\t=\t5.40\n");

    fprintf(genconf,"CANAL_longitudExterior\t=\t%f\n",CANAL_bond_reposo*2);

    fprintf(genconf,"CANAL_bond_fuerza\t=");
    for(i=0;i<N_simulaciones;i++)
        fprintf(genconf,"\t%.1f",CANAL_bond_fuerza_ini*pow(d_CANAL_bond_fuerza,i));
    fprintf(genconf,"\n");

    fprintf(genconf,"CANAL_bond_reposo\t=\t%.2f\n",CANAL_bond_reposo);

    fprintf(genconf,"CANAL_bending\t=");
    for(i=0;i<N_simulaciones;i++)
        fprintf(genconf,"\t%.1f",CANAL_bending_fuerza_ini*pow(d_CANAL_bending_fuerza,i));
    fprintf(genconf,"\n");

    fprintf(genconf,"CANAL_bending_ang\t=\t%.1f\n",CANAL_bending_ang);


    fclose(genconf);
}


///ANALISIS////////////////////////////////////////////////////////////////////
double CalculaRadioGiro(int index_sim_bond,int index_sim_bending, char* Carpeta, char* GROfile)
{
    int i,j;
    int N_particulas=0;
    double** Coordenadas;
    double RadioVector_medio[3]={0,0,0};
    double Delta_RadioVector[3];
    double radio_giro=0;
    char coord_data_name[256];
    char index_name[256];
    char seeker[256];
    char basura[256];
```

```
    char flag0='a', flag1='b';
    int indice_index;
    int indice_particula;
    FILE *coord_data;
    FILE * index;

    sprintf(index_name,"SISTEMA_index_%d_%d.ndx",index_sim_bond,index_sim_bending);
    index=fopen(index_name,"r");

    for(i=0;i<1;i++)
    {
        fscanf(index,"%s",seeker);

        if(strcmp(seeker,"[FUERA]")==0)
        {
            fseek(index,1,SEEK_CUR);
            for(j=0;j<1;j++)
            {
                if(N_particulas==0)
                    fscanf(index,"%d",&indice_index);
                else
                    fscanf(index,"%s",seeker);

                N_particulas++;
                flag0=fgetc(index);
                flag1=fgetc(index);
                fseek(index,-2,SEEK_CUR);
                if(flag1=='\n' || (flag1==EOF))
                    break;
                j=-1;
            }
            break;
        }
        i=-1;
    } //Aqui ya he contado cuantas particulas contribuyen para calcular el radio de giro

    Coordenadas=(double**)malloc(sizeof(double*)*N_particulas);
    for(i=0;i<N_particulas;i++)
        Coordenadas[i]=(double*)malloc(sizeof(double)*3);

    fclose(index);

    sprintf(coord_data_name,"%s/%s_%d_%d.gro",Carpeta,GROfile,index_sim_bond,index_sim_bending);
    coord_data=fopen(coord_data_name,"r");
    fscanf(coord_data,"%s%s",basura,basura);

    for(i=0;i<1;i++)
    {
        fscanf(coord_data,"%s%s%d%s%s%s%s%s",basura,basura,&indice_particula,basura,basura,basura,
            basura,basura,basura);
        if(indice_particula==indice_index-1)
        {
            for(j=0;j<N_particulas;j++)
                fscanf(coord_data,"%s%s%s%f%f%f%s%s%s",basura,basura,basura,&Coordenadas[j][0],&
                    Coordenadas[j][1],&Coordenadas[j][2],basura,basura,basura);
            break;
        }

        i=-1;
    } //Aqui ya estan almacenadas las coordenas para calcular el radio de giro


    for(i=0;i<N_particulas;i++)
    {
        RadioVector_medio[0]+=Coordenadas[i][0];
        RadioVector_medio[1]+=Coordenadas[i][1];
        RadioVector_medio[2]+=Coordenadas[i][2];
    }
    RadioVector_medio[0]=RadioVector_medio[0]/N_particulas;
    RadioVector_medio[1]=RadioVector_medio[1]/N_particulas;
    RadioVector_medio[2]=RadioVector_medio[2]/N_particulas;

    for(i=0;i<N_particulas;i++)
    {
        Delta_RadioVector[0]=Coordenadas[i][0]-RadioVector_medio[0];
        Delta_RadioVector[1]=Coordenadas[i][1]-RadioVector_medio[1];
        Delta_RadioVector[2]=Coordenadas[i][2]-RadioVector_medio[2];

        radio_giro+=ProductoEscalar(Delta_RadioVector,Delta_RadioVector);
    }
    radio_giro=sqrt(radio_giro/N_particulas);


    for(i=0;i<N_particulas;i++)
        free(Coordenadas[i]);
    free(Coordenadas);
    fclose(coord_data);

    return radio_giro;
}
```

```
void AnalisisDeModosNormales(struct VARIABLE_struct* VARIABLE_punt,struct CANAL_struct* CANAL_punt,
     int pinoffset,int ntmpi,int index_sim_bond,int index_sim_bending)
{
    int i;

    char comando[512];
    char CarpetaObjetivo[256];
    FILE *pipe;



    sprintf(CarpetaObjetivo,"Kb=%.2f_b0=%.2f_Ktheta=%.2f_theta0=%.2f__%d_%d",CANAL_punt->bond_fuerza
         [index_sim_bond],CANAL_punt->bond_reposo,CANAL_punt->bending_fuerza[index_sim_bending],
         CANAL_punt->bending_ang,index_sim_bond,index_sim_bending);


    if(fork()==0)
    {
        sprintf(comando,"mkdir %s\n",CarpetaObjetivo);

        pipe=popen(comando,"w");
        pclose(pipe);

        exit(0);
    }

    wait(NULL);


    if(fork()==0)
    {
        sprintf(comando,"cp ./Input/martini_mn.mdp ./Input/dry_martini_v2.1.itp SISTEMA_canal_%d_%d.
             itp SISTEMA_index_%d_%d.ndx topol_%d_%d.top AnalisisMN_min_%d_%d.gro traj_min_%d_%d.trr
              %s\n",index_sim_bond,index_sim_bending,index_sim_bond,index_sim_bending,index_sim_bond
             ,index_sim_bending,index_sim_bond,index_sim_bending,index_sim_bond,index_sim_bending,
             CarpetaObjetivo);

        pipe=popen(comando,"w");
        pclose(pipe);

        exit(0);
    }

    wait(NULL);



    if(fork()==0)
    {
        sprintf(comando,"cd ./%s\ngmx_d grompp -f martini_mn.mdp -c AnalisisMN_min_%d_%d.gro -t
             traj_min_%d_%d.trr -p topol_%d_%d.top -o topol_%d_%d.tpr -n SISTEMA_index_%d_%d.ndx -
             maxwarn 3\n",CarpetaObjetivo,index_sim_bond,index_sim_bending,index_sim_bond,
             index_sim_bending,index_sim_bond,index_sim_bending,index_sim_bond,index_sim_bending,
             index_sim_bond,index_sim_bending);

        pipe=popen(comando,"w");
        pclose(pipe);

        exit(0);
    }

    wait(NULL);


    if(fork()==0)
    {
        sprintf(comando,"cd ./%s\ngmx_d mdrun -s topol_%d_%d.tpr -c AnalisisMN_%d_%d.gro -mtx hess_%
             d_%d.mtx -rdd 2 -v -ntmpi %d -pin on -pinoffset %d -pinstride 1\n",CarpetaObjetivo,
             index_sim_bond,index_sim_bending,index_sim_bond,index_sim_bending,index_sim_bond,
             index_sim_bending,ntmpi,pinoffset);

        pipe=popen(comando,"w");
        pclose(pipe);

        exit(0);
    }

    wait(NULL);


    if(fork()==0)
    {
        sprintf(comando,"cd ./%s\ngmx_d nmeig -f hess_%d_%d.mtx -s topol_%d_%d.tpr -of eigenfreq_%d_
             %d.xvg -ol eigenval_%d_%d.xvg -os spectrum_%d_%d.xvg -v eigenvec_%d_%d.trr -first 1 -
             last %d\n",CarpetaObjetivo,index_sim_bond,index_sim_bending,index_sim_bond,
             index_sim_bending,index_sim_bond,index_sim_bending,index_sim_bond,index_sim_bending,
             index_sim_bond,index_sim_bending,index_sim_bond,index_sim_bending,3*CANAL_punt->n);

        pipe=popen(comando,"w");
        pclose(pipe);

        exit(0);
```

```
}

wait(NULL);

/*if(fork()==0)
{
     sprintf(comando,"cd ./%s\ngmx_d anaeig -v eigenvec_%d_%d.trr -s topol_%d_%d.tpr -n
          SISTEMA_index_%d_%d.ndx -eig eigenval_%d_%d.xvg -comp eigcomp_%d_%d.xvg -first 1 -last
          -1 -xvg none\n",CarpetaObjetivo,index_sim_bond,index_sim_bending,index_sim_bond,
          index_sim_bending,index_sim_bond,index_sim_bending,index_sim_bond,index_sim_bending,
          index_sim_bond,index_sim_bending);

     pipe=popen(comando,"w");
     pclose(pipe);

     exit(0);
}
wait(NULL);*/




for(i=7;i<=3*VARIABLE_punt->SISTEMA_n;i++)
{
     if(fork()==0)
     {
          sprintf(comando,"cd ./%s\ngmx_d nmtraj -s topol_%d_%d.tpr -v eigenvec_%d_%d.trr -o
               autovector_%d__%d_%d.gro -eignr %d -temp 50000 -phases 90.0 -nframes 1 \n",
               CarpetaObjetivo,index_sim_bond,index_sim_bending,index_sim_bond,index_sim_bending,i
               ,index_sim_bond,index_sim_bending,i);

          pipe=popen(comando,"w");
          pclose(pipe);

          exit(0);
     }
     wait(NULL);

     if(fork()==0)
     {
          sprintf(comando,"cp ./%s/autovector_%d__%d_%d.gro ./\n",CarpetaObjetivo,i,index_sim_bond
               ,index_sim_bending);

          pipe=popen(comando,"w");
          pclose(pipe);

          exit(0);
     }
     wait(NULL);
}

if(fork()==0)
{
     sprintf(comando,"cp ./%s/eigenfreq_%d_%d.xvg ./\n",CarpetaObjetivo,index_sim_bond,
          index_sim_bending);

     pipe=popen(comando,"w");
     pclose(pipe);

     exit(0);
}
wait(NULL);







     if(fork()==0)
     {
          sprintf(comando,"rm -r Kb=%.2f_b0=%.2f_Ktheta=%.2f_theta0=%.2f__%d_%d\n",CANAL_punt->
               bond_fuerza[index_sim_bond],CANAL_punt->bond_reposo,CANAL_punt->bending_fuerza[
               index_sim_bending],CANAL_punt->bending_ang,index_sim_bond,index_sim_bending);

          pipe=popen(comando,"w");
          pclose(pipe);

          exit(0);
     }

     wait(NULL);




if(fork()==0)
{
     sprintf(comando,"rm SISTEMA_canal_%d_%d.itp SISTEMA_index_%d_%d.ndx topol_%d_%d.top
          AnalisisMN_min_%d_%d.gro traj_min_%d_%d.trr\n",index_sim_bond,index_sim_bending,
          index_sim_bond,index_sim_bending,index_sim_bond,index_sim_bending,index_sim_bond,
```

```
                index_sim_bending,index_sim_bond,index_sim_bending);

            pipe=popen(comando,"w");
            pclose(pipe);

            exit(0);
        }

        wait(NULL);


}



///LEER INPUT
        ////////////////////////////////////////////////////////////////////////////////////
void LeerInput(struct CANAL_struct* CANAL_punt,struct VARIABLE_struct* VARIABLE_punt)
{
        int i,j;
        FILE *input;
        char basura[256];

        input=fopen("./Input/genconf.txt","r");

        fscanf(input,"%s %s %d\n",basura,basura,&VARIABLE_punt->N_simulaciones);

        CANAL_punt->bond_fuerza=(double*)malloc(sizeof(double)*(VARIABLE_punt->N_simulaciones));

        CANAL_punt->bending_fuerza=(double*)malloc(sizeof(double)*(VARIABLE_punt->N_simulaciones));

        fscanf(input,"%s\t%s\t%s\n",basura,basura,CANAL_punt->Borde_poro);
        fscanf(input,"%s\t%s\t%s\n",basura,basura,CANAL_punt->Cuerpo_poro);


        fscanf(input,"%s %s %lf",basura,basura,&CANAL_punt->radio);

        fscanf(input,"%s %s %lf",basura,basura,&CANAL_punt->longitudInterior);


        fscanf(input,"%s %s %lf",basura,basura,&CANAL_punt->longitudExterior);


        fscanf(input,"%s %s",basura,basura);
        for(i=0;i<VARIABLE_punt->N_simulaciones;i++)
                fscanf(input,"%lf",&CANAL_punt->bond_fuerza[i]);

        fscanf(input,"%s %s %lf",basura,basura,&CANAL_punt->bond_reposo);


        fscanf(input,"%s %s",basura,basura);
        for(i=0;i<VARIABLE_punt->N_simulaciones;i++)
                fscanf(input,"%lf",&CANAL_punt->bending_fuerza[i]);


        fscanf(input,"%s %s %lf",basura,basura,&CANAL_punt->bending_ang);


        fclose(input);

}



///LIBERACION DE MEMORIA Y LIMPIEZA////////////////////////////////////
void LiberarMemoriaReservada(struct CANAL_struct* CANAL_punt,struct VARIABLE_struct* VARIABLE_punt)
{
        int i;
        free(CANAL_punt->bond_fuerza);

        free(CANAL_punt->bending_fuerza);
}




/*************************
 *                       *
 * FUNCIONES ESPECIFICAS *
 *                       *
 *************************/

void GeneraMDP(struct VARIABLE_struct* VARIABLE_punt)
{
        FILE *min,*sim,*dry_martini;



        min=fopen("Input/minim.mdp","w");
        sim=fopen("Input/martini_mn.mdp","w");
```

```
    dry_martini=fopen("Input/dry_martini_v2.1.itp","w");

    fprintf(min,"integrator\t= cg\n\nemstep\t\t = 0.001\nemtol\t\t= 0.01\nnsteps\t\t= 500000\ncomm-
        mode\t= Linear\ncomm-grps\t= \nnstcgsteep\t\t= 1000\n\nnstlist\t\t = 1\nnstxout\t\t = 0\
        nnstvout\t\t = 0\nnstfout\t\t = 0\nnstlog\t\t = 0\nnstenergy\t\t = 0\nnstxout-compressed\t\
        t = 0\ncutoff-scheme\t\t = verlet \nverlet-buffer-drift\t= 0.005\nns_type\t\t = grid\n\
        coulombtype\t\t = reaction-field\n\nrcoulomb\t\t = 1.2\n\nvdw-type\t\t = cutoff\nvdw-
        modifier\t\t = Potential-shift-verlet\nrvdw\t\t = 1.2\n\npbc\t\t = xyz\n");



    fprintf(sim,"\ntitle\t\t = Analisis_Modos_Normales\n\n\n\nintegrator\t\t = nm\n\ncutoff-scheme
        \t\t = Verlet\nstlist\t\t = 10\ns_type\t\t = grid\npbc\t\t = xyz\nverlet-buffer-tolerance
        \t = 0.005\n\n\coulombtype\t\t = reaction-field\rcoulomb\t\t = 1.1\epsilon_r\t\t = 15\t\
        n\vdw_type\t\t = cutoff \nvdw-modifier\t\t = Potential-shift-verlet\nrvdw\t\t = 1.1\t\n\
        nconstraints\t\t = none \nconstraint_algorithm\t\t = Lincs\n");



    fprintf(dry_martini,"[dry_martini]");


    fclose(min);
    fclose(sim);
    fclose(dry_martini);
}



void MinimizaEstructura(struct VARIABLE_struct* VARIABLE_punt,struct CANAL_struct* CANAL_punt,int
     pinoffset,int ntmpi, int index_sim_bond,int index_sim_bending)
{
    double box_x;
    double box_y;
    double box_z;
    char comando[512];
    FILE *pipe;

    if(fork()==0)
    {
        sprintf(comando,"mkdir CarpetaPrincipal_%d_%d\n",index_sim_bond,index_sim_bending);

        pipe=popen(comando,"w");
        pclose(pipe);

        exit(0);
    }
    wait(NULL);

    box_x=3+CANAL_punt->longitudInterior+CANAL_punt->longitudExterior;
    box_y=box_x;
    box_z=box_x;

    if(fork()==0)
    {
        sprintf(comando,"gmx_d editconf -f SISTEMA_conf_%d_%d.gro -o Sistema_%d_%d.gro -box %.2f %.2
            f %.2f -c\n",index_sim_bond,index_sim_bending,index_sim_bond,index_sim_bending,box_x,
            box_y,box_z);

        pipe=popen(comando,"w");
        pclose(pipe);

        exit(0);
    }
    wait(NULL);


    EscribeFicheroTopol(index_sim_bond,index_sim_bending);


    if(fork()==0)
    {
        sprintf(comando,"cp ./Input/minim.mdp ./Input/term.mdp ./Input/dry_martini_v2.1.itp Sistema_
            %d_%d.gro SISTEMA_canal_%d_%d.itp SISTEMA_index_%d_%d.ndx topol_%d_%d.top
            CarpetaPrincipal_%d_%d\n",index_sim_bond,index_sim_bending,index_sim_bond,
            index_sim_bending,index_sim_bond,index_sim_bending,index_sim_bond,index_sim_bending,
            index_sim_bond,index_sim_bending);

        pipe=popen(comando,"w");
        pclose(pipe);

        exit(0);
    }
    wait(NULL);

    if(fork()==0)
    {
        sprintf(comando,"cd CarpetaPrincipal_%d_%d\ngmx_d grompp -f minim.mdp -c Sistema_%d_%d.gro -
            p topol_%d_%d.top -o topol_%d_%d.tpr -n SISTEMA_index_%d_%d.ndx -maxwarn 3\n",
            index_sim_bond,index_sim_bending,index_sim_bond,index_sim_bending,index_sim_bond,
            index_sim_bending,index_sim_bond,index_sim_bending,index_sim_bond,index_sim_bending);
```

```
        pipe=popen(comando,"w");
        pclose(pipe);

        exit(0);
    }
    wait(NULL);


    if(fork()==0)
    {
        sprintf(comando,"cd CarpetaPrincipal_%d_%d\ngmx_d mdrun -s topol_%d_%d.tpr -c
            AnalisisMN_min_%d_%d.gro -o traj_min_%d_%d.trr -rdd 2 -v -ntmpi %d -pin on -pinoffset %
            d -pinstride 1\n",index_sim_bond,index_sim_bending,index_sim_bond,index_sim_bending,
            index_sim_bond,index_sim_bending,index_sim_bond,index_sim_bending,ntmpi,pinoffset);

        pipe=popen(comando,"w");
        pclose(pipe);

        exit(0);
    }
    wait(NULL);



    if(fork()==0)
    {
        sprintf(comando,"cp CarpetaPrincipal_%d_%d/AnalisisMN_min_%d_%d.gro CarpetaPrincipal_%d_%d/
            SISTEMA_index_%d_%d.ndx CarpetaPrincipal_%d_%d/traj_min_%d_%d.trr ./\n",index_sim_bond,
            index_sim_bending,index_sim_bond,index_sim_bending,index_sim_bond,index_sim_bending,
            index_sim_bond,index_sim_bending,index_sim_bond,index_sim_bending,index_sim_bond,
            index_sim_bending);

        pipe=popen(comando,"w");
        pclose(pipe);

        exit(0);
    }
    wait(NULL);


    if(fork()==0)
    {
        sprintf(comando,"rm Sistema_%d_%d.gro SISTEMA_conf_%d_%d.gro\nrm -r CarpetaPrincipal_%d_%d\n
            ",index_sim_bond,index_sim_bending,index_sim_bond,index_sim_bending,index_sim_bond,
            index_sim_bending);

        pipe=popen(comando,"w");
        pclose(pipe);

        exit(0);
    }
    wait(NULL);


}



void ProcesoResultados(struct CANAL_struct* CANAL_punt,struct VARIABLE_struct* VARIABLE_punt,int
    index_sim_bond,int index_sim_bending)
{
    int i,j,k,x,x_bueno;;
    double y;
    char flag;
    char comando[512];
    FILE *freq,*Nfreq;
    FILE *pipe;
    char freq_name[256];
    char Nfreq_name[256];

    sprintf(freq_name,"eigenfreq_%d_%d.xvg",index_sim_bond,index_sim_bending);
    sprintf(Nfreq_name,"eigenfreq_%d_%d.txt",index_sim_bond,index_sim_bending);

    x_bueno=AnalisisDeAutovectores(VARIABLE_punt,CANAL_punt,index_sim_bond,index_sim_bending);

    freq=fopen(freq_name,"r");
    Nfreq=fopen(Nfreq_name,"w");

    for(i=0;i<1;i++)
    {
        flag=fgetc(freq);
        if(flag=='@' || flag=='#' || flag=='&')
        {
            for(k=0;k<1;k++)
            {
                if(fgetc(freq)=='\n')
                    break;
                k=-1;
            }
        }
        else
```

```
        {
            for(j=0;j<1;j++)
            {
                fscanf(freq,"%d %f",&x,&y);
                if(x==x_bueno)
                    fprintf(Nfreq,"%d\t%f\n",x,y*VelocidadLuz_c);
                if(feof(freq)!=0)
                    break;
                j=-1;
            }
            break;
        }
        i=-1;
    }
    fclose(freq);
    fclose(Nfreq);


    if(fork()==0)
    {
        sprintf(comando,"rm %s\n",freq_name);

        pipe=popen(comando,"w");
        pclose(pipe);

        exit(0);
    }
    wait(NULL);

}


void AnalisisDeResultados(struct VARIABLE_struct* VARIABLE_punt, struct CANAL_struct* CANAL_punt)
{
    int i,index_sim_bond,index_sim_bending,x;
    double y;
    char eigenfreq_name[256];
    char comando[512];
    char flag;
    FILE *pipe;
    FILE *eigenfreq;
    FILE *MapaFrec;


    MapaFrec=fopen("Mapa_Frecuencias.txt","w");
    fprintf(MapaFrec,"K_b\tK_theta\tmodo\tFreq[ps^-1]\n");


    for(index_sim_bond=0;index_sim_bond<VARIABLE_punt->N_simulaciones;index_sim_bond++)
    {
        for(index_sim_bending=0;index_sim_bending<VARIABLE_punt->N_simulaciones;index_sim_bending++)
        {
            sprintf(eigenfreq_name,"eigenfreq_%d_%d.txt",index_sim_bond,index_sim_bending);

            eigenfreq=fopen(eigenfreq_name,"r");

            fscanf(eigenfreq,"%d %f",&x,&y);

            fprintf(MapaFrec,"%f\t%f\t%d\t%f\n",CANAL_punt->bond_fuerza[index_sim_bond],
                CANAL_punt->bending_fuerza[index_sim_bending],x,y);
            fflush(MapaFrec);


            fclose(eigenfreq);
        }
    }

    fclose(MapaFrec);



    for(index_sim_bond=0;index_sim_bond<VARIABLE_punt->N_simulaciones;index_sim_bond++)
    {
        for(index_sim_bending=0;index_sim_bending<VARIABLE_punt->N_simulaciones;index_sim_bending++)
        {
            if(fork()==0)
            {
                sprintf(comando,"rm eigenfreq_%d_%d.txt\n",index_sim_bond,index_sim_bending);

                pipe=popen(comando,"w");
                pclose(pipe);

                exit(0);
            }
            wait(NULL);
        }

    }
}


int AnalisisDeAutovectores(struct VARIABLE_struct* VARIABLE_punt, struct CANAL_struct* CANAL_punt,
    int index_sim_bond,int index_sim_bending)
```

```
{
    int i;
    int index_modo;
    int resultado=0;
    char basura[256];
    char inst_name[256];
    char comando[1024];
    char flag;
    double x,y,z;
    double producto=0,producto_previo=0,norma_ref=0,norma_inst=0;
    double *Autocutoff;
    FILE *referencia,*instantaneo,*cutoff,*pipe;

    VARIABLE_punt->ModoNormal=(double**)malloc(sizeof(double*)*(3*VARIABLE_punt->SISTEMA_n+1));
    for(i=0;i<(3*VARIABLE_punt->SISTEMA_n+1);i++)
        VARIABLE_punt->ModoNormal[i]=(double*)malloc(sizeof(double)*3*VARIABLE_punt->SISTEMA_n);

    CANAL_punt->Nb=(int)((CANAL_punt->longitudInterior+2*CANAL_punt->longitudExterior)/CANAL_punt->
        bond_reposo);
    CANAL_punt->M=(int)(2*PI/(acos(1-CANAL_punt->bond_reposo*CANAL_punt->bond_reposo/(2*CANAL_punt->
        radio*CANAL_punt->radio))));
    VARIABLE_punt->SISTEMA_n=CANAL_punt->Nb*CANAL_punt->M;

    Autocutoff=(double*)malloc(sizeof(double)*3*VARIABLE_punt->SISTEMA_n);

    cutoff=fopen("ModoCutoff.txt","r");
    for(i=0;i<1;i++)
    {
        flag=fgetc(cutoff);
        if(flag=='\n')
            break;
        i=-1;
    }
    fscanf(cutoff,"%s",basura);

    for(i=0;i<3*VARIABLE_punt->SISTEMA_n;i+=3)
    {
        fscanf(cutoff,"%s %s %s",basura,basura,basura);
        fscanf(cutoff,"%lf %lf %lf",&x,&y,&z);

        Autocutoff[i]=x;
        Autocutoff[i+1]=y;
        Autocutoff[i+2]=z;
    }
    fclose(cutoff);


    referencia=fopen("ModoReferencia.txt","r");
    for(i=0;i<1;i++)
    {
        flag=fgetc(referencia);
        if(flag=='\n')
            break;
        i=-1;
    }
    fscanf(referencia,"%s",basura);

    for(i=0;i<3*VARIABLE_punt->SISTEMA_n;i+=3)
    {
        fscanf(referencia,"%s %s %s",basura,basura,basura);
        fscanf(referencia,"%lf %lf %lf",&x,&y,&z);

        VARIABLE_punt->ModoNormal[0][i]=x-Autocutoff[i];
        VARIABLE_punt->ModoNormal[0][i+1]=y-Autocutoff[i+1];
        VARIABLE_punt->ModoNormal[0][i+2]=z-Autocutoff[i+2];
    }
    fclose(referencia);


    for(index_modo=7;index_modo<=3*VARIABLE_punt->SISTEMA_n;index_modo++)
    {
        sprintf(inst_name,"autovector_%d__%d_%d.gro",index_modo,index_sim_bond,index_sim_bending);
        instantaneo=fopen(inst_name,"r");
        for(i=0;i<1;i++)
        {
            flag=fgetc(instantaneo);
            if(flag=='\n')
                break;
            i=-1;
        }
        fscanf(instantaneo,"%s",basura);

        for(i=0;i<3*VARIABLE_punt->SISTEMA_n;i+=3)
        {
            fscanf(instantaneo,"%s %s %s",basura,basura,basura);
            fscanf(instantaneo,"%lf %lf %lf",&x,&y,&z);

            VARIABLE_punt->ModoNormal[index_modo][i]=x-Autocutoff[i];
            VARIABLE_punt->ModoNormal[index_modo][i+1]=y-Autocutoff[i+1];
            VARIABLE_punt->ModoNormal[index_modo][i+2]=z-Autocutoff[i+2];
        }
        fclose(instantaneo);
```

```
        if(fork()==0)
        {
            sprintf(comando,"rm %s\n",inst_name);
            pipe=popen(comando,"w");
            pclose(pipe);

            exit(0);
        }
        wait(NULL);

        //OrientarAutovector(VARIABLE_punt,CANAL_punt,VARIABLE_punt->ModoNormal[0],VARIABLE_punt->
            ModoNormal[index_modo]);
    }



    norma_ref=0;
    for(i=0;i<3*VARIABLE_punt->SISTEMA_n;i++)
        norma_ref+=VARIABLE_punt->ModoNormal[0][i]*VARIABLE_punt->ModoNormal[0][i];
    norma_ref=sqrt(norma_ref);

    for(index_modo=7;index_modo<=50;index_modo++)
    {
        producto=0;
        norma_inst=0;
        for(i=0;i<3*VARIABLE_punt->SISTEMA_n;i++)
        {
            producto+=VARIABLE_punt->ModoNormal[0][i]*VARIABLE_punt->ModoNormal[index_modo][i];
            norma_inst+=VARIABLE_punt->ModoNormal[index_modo][i]*VARIABLE_punt->ModoNormal[
                index_modo][i];
        }
        //norma_inst=sqrt(norma_inst);
        producto=producto/(norma_ref*norma_ref);

        //printf("\nindex_modo= %d\tproducto=%lf\n",index_modo,producto);
        if(producto>producto_previo)
        {
            //printf("\n%lf>%lf\n",producto,producto_previo);
            producto_previo=producto;
            resultado=index_modo;
            //printf("\nresultado= %d\n",resultado); getchar();
        }
    }

    for(i=0;i<(3*VARIABLE_punt->SISTEMA_n+1);i++)
        free(VARIABLE_punt->ModoNormal[i]);
    free(VARIABLE_punt->ModoNormal);

    free(Autocutoff);

    return resultado;
}


void OrientarAutovector(struct VARIABLE_struct* VARIABLE_punt, struct CANAL_struct* CANAL_punt,
    double *referencia,double* Autovector)
{
    int i;
    double eje_z[3]={0,0,1.0};
    double **particula;
    double **particula_ref;
    double *angulos_ref;
    double *angulos;

    CANAL_punt->Nb=(int)((CANAL_punt->longitudInterior+2*CANAL_punt->longitudExterior)/CANAL_punt->
        bond_reposo);
    CANAL_punt->M=(int)(2*PI/(acos(1-CANAL_punt->bond_reposo*CANAL_punt->bond_reposo/(2*CANAL_punt->
        radio*CANAL_punt->radio))));
    VARIABLE_punt->SISTEMA_n=CANAL_punt->Nb*CANAL_punt->M;

    particula=(double**)malloc(sizeof(double*)*VARIABLE_punt->SISTEMA_n);
    for(i=0;i<VARIABLE_punt->SISTEMA_n;i++)
        particula[i]=(double*)malloc(sizeof(double)*3);

    particula_ref=(double**)malloc(sizeof(double*)*VARIABLE_punt->SISTEMA_n);
    for(i=0;i<VARIABLE_punt->SISTEMA_n;i++)
    particula_ref[i]=(double*)malloc(sizeof(double)*3);

    angulos=(double*)malloc(sizeof(double)*VARIABLE_punt->SISTEMA_n);

    angulos_ref=(double*)malloc(sizeof(double)*VARIABLE_punt->SISTEMA_n);

    for(i=0;i<3*VARIABLE_punt->SISTEMA_n;i=i+3)
    {
        particula_ref[i/3][0]=referencia[i];
        particula_ref[i/3][1]=referencia[i+1];
        particula_ref[i/3][2]=referencia[i+2];

        particula[i/3][0]=Autovector[i];
        particula[i/3][1]=Autovector[i+1];
        particula[i/3][2]=Autovector[i+2];
    }
```

```
    for(i=0;i<VARIABLE_punt->SISTEMA_n;i++)
    {
        angulos_ref[i]=atan(particula_ref[i][1]/particula_ref[i][0]);
        angulos[i]=atan(particula[i][1]/particula[i][0]);
    }

    for(i=0;i<VARIABLE_punt->SISTEMA_n;i++)
        Rotacion(eje_z,particula[i],angulos_ref[i]-angulos[i]);


    for(i=0;i<3*VARIABLE_punt->SISTEMA_n;i=i+3)
    {
        Autovector[i]=particula[i/3][0];
        Autovector[i+1]=particula[i/3][1];
        Autovector[i+2]=particula[i/3][2];
    }

    for(i=0;i<VARIABLE_punt->SISTEMA_n;i++)
        free(particula[i]);
    free(particula);

    for(i=0;i<VARIABLE_punt->SISTEMA_n;i++)
    free(particula_ref[i]);
    free(particula_ref);

    free(angulos_ref);

    free(angulos);
}
```