Transición Sólido-Líquido en un sistema de Lennard-Jones

Marcos Román

December 19, 2011

Chapter 1

Introducción teórica

1.1 Dinámica molecular

La dinámica molecular es una poderosa técnica de simulación utilizada para estudiar las propiedades físicas y químicas de sólidos, líquidos, materiales amorfos, moléculas en biologia, etc.

En la versión más común (utilizada en este trabajo) las trayectorias de las partículas se obtienen resolviendo numéricamente las ecuaciones de movimiento de Newton, donde las fuerzas estan dadas a partir de la energía potencial entre pares de partículas.

Éstas interactúan durante periodos de tiempo finitos y permiten el cálculo de distintas propiedades y características de sistemas que resultan imposibles de obtener analíticamente; en dinámica molecular se sortea esta barrera mediante el empleo de aproximaciones numéricas. Los resultados se basan en la hipótesis ergódica: los promedios estadísticos del ensamble son iguales a los promedios temporales del sistema.

Dado que un sistema de moleculas es dinámico, las velocidades y posiciones de las moleculas cambian continuamente, asi que será necesario seguir el movimiento de cada molécula a cada tiempo para determinar sus efectos sobre otras moleculas, que estan también en movimiento. Luego de que la simulación corra por un tiempo suficiente y el sistema en cuestión se estabilice, se calculan los promedios temporales de las cantidades dinámicas para deducir las propiedades termodinámicas. Aplicamos las leyes de Newton asumiendo que la fuerza neta en cada molécula es la suma de fuerzas de a pares con las demas N-1 moléculas:

$$m\frac{d^2\mathbf{r}_i}{dt^2} = \mathbf{F}_i(\mathbf{r}_1, \ldots, \mathbf{r}_N) \tag{1.1}$$

$$m\frac{d^2\mathbf{r}_i}{dt^2} = \sum_{i<j=0}^{N} \mathbf{f}_{ij}, \quad i = 1, \ldots, N \tag{1.2}$$

La fuerza sobre la partícula i-ésima se deriva a partir de un potencial de interacción de a pares,

$$\mathbf{F}_i(\mathbf{r}_1, \ldots, \mathbf{r}_N) = \nabla_{\mathbf{r}_i} U(\mathbf{r}_1, \ldots, \mathbf{r}_N) \tag{1.3}$$

$$U(\mathbf{r}_1, \ldots, \mathbf{r}_N) = \sum_{i<j} u(r_{ij}) \tag{1.4}$$

Como el potencial utilizado es conservativo, la energía total del sistema ($E_{total} = E_{cinetica} + E_{potencial}$) debería mantenerse constante. Sin embargo, en una simulación ahorramos recursos computacionales introduciendo un radio de corte; es decir, manteniendo la interacción solo para partículas con distancias menores a r_{max}. Esto produce una discontinuidad, por lo que la derivada del potencial es infinita en este punto y la energía ya no será conservada. Sin embargo, r_{max} suele escogerse donde las fuerzas de interacción son minúsculas, por lo que la violación de la conservación de la energía debería ser pequeña en comparación a los errores de redondeo y otras aproximaciones.

1.2 Potencial de Lennard-Jones

El potencial que utilizaremos es el potencial de Lennard-Jones.

$$u(r) = 4\epsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right] \tag{1.5}$$

Este puede puede verse como la suma de una interacción atractiva de largo alcance proporcional a $1/r^6$ y una repulsiva de corto alcance $1/r^{12}$. El cambio de repulsión a atracción ocurre para $r = \sigma$. El mínimo del potencial a $r = 2^{1/6}\sigma = 1.123\sigma$.

Resulta útil utilizar unidades naturales para las variables, de forma tal que las constantes que aparecen sean iguales a la unidad en tales unidades.

De esta forma el potencial es,

$$u(r) = 4 \left[\left(\frac{1}{r}\right)^{12} - \left(\frac{1}{r}\right)^{6} \right] \tag{1.6}$$

1.3 Conexión con variables termodinámicas

Asumiendo que el número de partículas es suficientemente grande, hacemos uso de los resultados de mecánica estadística para relacionar los resultados de nuestra simulación a las cantidades termodinámicas. El teorema de equipartición, por ejemplo, nos dice que para las moléculas en equilibrio térmico a temperatura T, cada grado de libertad molecular tiene una energía media $\frac{1}{2}k_BT$ asociada, donde k_B es la constante de Boltzmann ($1.38 \times 10^{-23} J/K$). La simulación nos provee de la energía cinetica media.

$$E_{cinetica} = \sum_{i=1}^{N} \frac{1}{2} \|\mathbf{v}_i\|^2 \tag{1.7}$$

El promedio de esta cantidad se relaciona con la temperatura mediante

$$E_{cinetica} = \frac{3}{2} N k_B T \tag{1.8}$$

La presión del sistema se determina mediante una versión del teorema del virial,

$$PV = Nk_BT + w/3 \tag{1.9}$$

$$w = \sum_{i<j} \mathbf{r}_{ij} \cdot \mathbf{f}_{ij} \tag{1.10}$$

donde el virial w es una suma pesada de fuerzas.

1.4 Condiciones Periódicas de Contorno

Un ordenador cuenta con una cantidad finita de memoria que puede almacenar, por lo que se halla limitado también el tamaño del sistema que resulta posible simular. Lo más simple y controlable resulta ser tener a las partículas en una caja rectangular y hacer uso de coordenadas cartesianas para describir las posiciones y velocidades, que es lo que comúnmente se hace. Sin embargo, tal sistema se halla lejos de comportarse de manera similar a la que se vería en el límite termodinámico porque se introducen inevitablemente efectos de superficie artificiales. Para resolver este problema que se presenta al contar con un número reducido de partículas, imponemos condiciones periódicas de contorno; esto es, nos imaginamos que la caja en la que se hallan las partículas se repite indefinidamente en todas las direcciones. De esta forma, para cada paso de simulación examinamos la posición de las particulas y vemos si han salido de la región de simulación establecida. Si esto ocurre, entonces llevamos la partícula al lado opuesto de la caja, es decir (considerando una dimensión):

$$x \to \begin{cases} x + L & \text{si } x \leq 0 \\ x - L & \text{si } x > L \end{cases} \tag{1.11}$$

De acuerdo con esto, cada caja se ve de la misma forma y tiene propiedades continuas en los bordes. Así también, se mantiene constante siempre el número de partículas.

1.5 Algoritmo de Verlet

El algoritmo de Verlet consiste en un empleo de la denominada "aproximación forward" de la derivada para avanzar la posición y velocidad de cada partícula simultáneamente en cada paso de simulación.

$$\mathbf{r}_i(t+h) = \mathbf{r}_i(t) + h\mathbf{v}_i(t) + \frac{h^2}{2}\mathbf{F}_i(t) + O(h^3) \tag{1.12}$$

$$\mathbf{v}_i(t+h) = \mathbf{v}_i(t) + h\left[\frac{\mathbf{F}_i(t+h) + \mathbf{F}_i(t)}{2}\right] + \frac{h^2}{2}\mathbf{F}_i(t) + O(h^2) \tag{1.13}$$

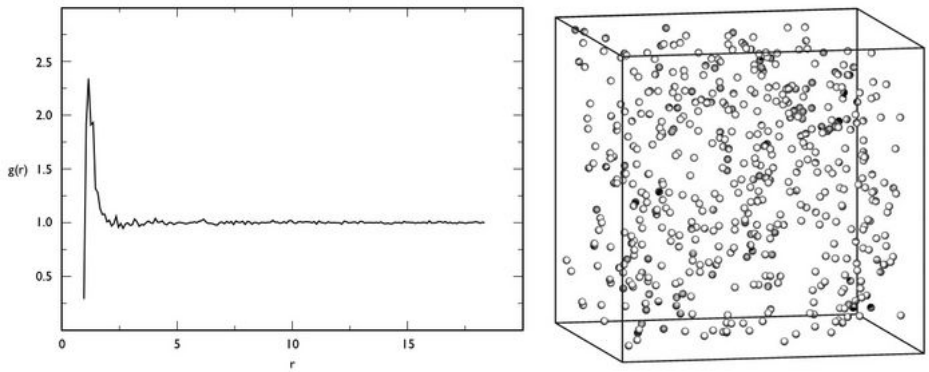
1.6 Función de distribución radial

La función de distribución radial, conocida también como FDR o función de correlación de pares, permite cuantificar la correlación entre partículas de un sistema. Específicamente mide, en promedio, la probabilidad de hallar una partícula a una distancia r de una partícula de referencia, relativa a la del gas ideal. Es de importancia fundamental en termodinámica ya que se pueden calcular propiedades macroscopicas del sistema mediante ella, por ejemplo la energía, presión o compresibilidad de un fluido. Esta funcion se suele denotar por $g(r)$. $\rho g(r)dr$ nos da la probabilidad de hallar una partícula a distancia r dado que tenemos un átomo en el origen. Esta sin embargo no esta normalizada y tendremos que

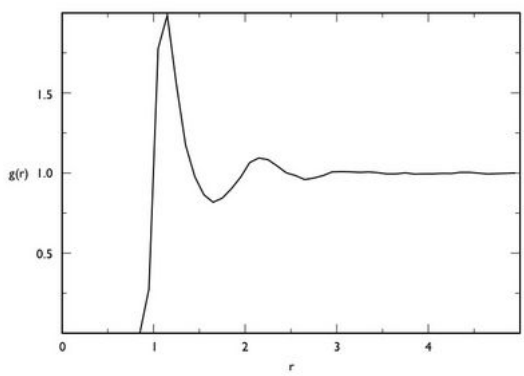
$$\int_0^\infty \rho g(r)dr = N - 1 \tag{1.14}$$

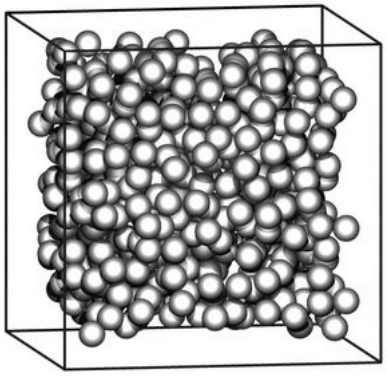
Veamos la forma que debería tener la FDR cuando la fase de un sistema de partículas se halla bien definida.

En el primer caso, si tenemos un gas de Lennard-Jones (500 partículas, $\rho = 0.01$, $T = 1$) $g(r)$ será, tal como se espera, aproximadamente igual en forma a la FDR de un gas ideal.

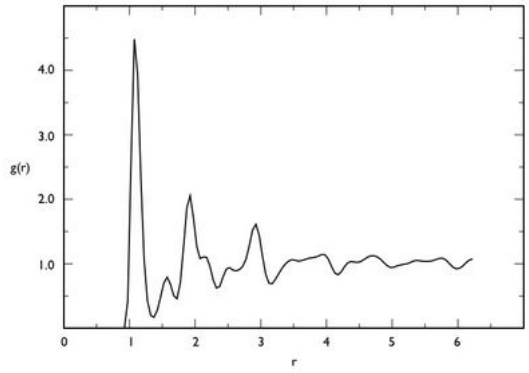


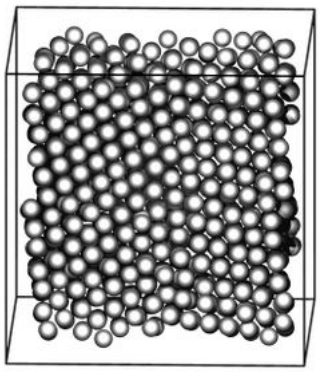
En el caso de un líquido (500 partículas, $\rho = 0.5$, $T = 1$)





Finalmente, para un sólido (2000 partículas, $\rho = 1$, $T = 0.5$):





Chapter 2

Problema a resolver

Se busca determinar, para diferentes densidades y temperaturas, el estado en el que se halla un sistema conformado por 864 partículas que interactúan entre sí de a pares mediante el potencial de Lennard-Jones. Se explora la zona que debería corresponder, segun otros estudios (ver paper de Hansen-Verlet) a la transición fluido-sólido, por lo cual se espera hallar coexistencia entre estas fases o bien fases puras.

Chapter 3

Metodología

Simularemos el sistema en el ensamble microcanonico (NVE), utilizando los metodos de dinamica molecular. Iniciamos cada simulación con las partículas dispuestas en un cristal FCC, lo cual corresponde a la estructura cristalina que presenta el sistema de Lennard-Jones en su fase sólida. Utilizamos $864 = 4 \times 6^3$ partículas e imponemos condiciones periodicas de contorno, mediante las cuales se logra un comportamiento del sistema finito más parecido al que se tendría en el límite termodinámico. Ya al iniciar el sistema tenemos fijada la densidad (que luego solo varía localmente en ciertos casos, pero la relación N/V se mantiene constante). Para explorar el diagrama de fases, deseamos que el sistema adquiera una temperatura (promedio) estable, por lo cual antes de proceder a determinar de qué fase se trata, dadas las coordenadas (densidad,temperatura) en el diagrama de fases, es necesario primeramente termalizar el sistema para obtener el estado de equilibrio deseado. Este proceso de termalización consiste en reescalear (multiplicar por algun factor constante) las velocidades de todas las particulas. Para determinar esta constante nos servimos de la ecuación (1.7) que relaciona la temperatura con la energía cinética de las partículas. En total, se toman $N_{termalizacion}$ pasos de termalización, en los cuales reescaleamos la velocidad cada N_{tcada} pasos.

Una vez termalizado el sistema, procedemos a calcular lo siguiente: temperatura (promedio de temperatura instantánea), presión promedio, función de distribución radial y parámetro de orden medio. A todo esto lo calculamos (o muestreamos) durante $N_{evolucion}$ pasos siguientes luego de la termalización, con lo que en total hacemos para cada simulación $N_{total} = N_{termalizacion} + N_{evolucion}$ pasos.

De lo calculado, determinamos la fase del sistema de acuerdo al valor del parámetro de orden y la forma de la FDR.

El parámetro de orden utilizado

$$S = \sum_{i=1}^{N} \| e^{i\mathbf{k}\cdot\mathbf{r}_i} \|^2 \tag{3.1}$$

en donde (l es el lado de la celda)

$$\mathbf{k} = \frac{2\pi}{l}(-1, 1, -1) \tag{3.2}$$

nos daría 1 en caso de tener la configuración FCC inicial; esto no se observa realmente dado que la posición de cada particula fluctúa alrededor de la de equilibrio o que incluso el sistema, como un todo, puede desplazarse aun formando un cristal FCC. De todas formas, lo importante es que para un sistema desordenado como un gas este parametro es casi nulo (en nuestros resultados, del orden de 10^{-4}) mientras que para un sistema ordenado es mayor, llegando a valores del orden de la unidad.

La función de distribución radial, así también, presenta formas muy distintas de acuerdo a la fase en la que se halla el sistema, o dicho de otra forma, a las posiciones relativas medias entre las partículas. Lo que cuantifica $g(r)$ es precisamente esto; nos da una idea de cómo se ve el sistema en promedio si uno se situa en donde hay una partícula y mira alrededor. La función se obtiene considerando por igual la contribución de todas las partículas.

En un cristal perfecto (con simetría traslacional y a temperatura $T = 0$) se tiene desde cada posición exactamente el mismo panorama; cada partícula se halla a una distancia fija y la función de distribución

radial tendrá entonces muchos picos a distancias precisas. Para valores de r intermedios tendremos g(r)=0 ya que no habra ningun par de particulas que se hallen a tales distancias. Si incluimos el efecto de las fluctuaciones termicas, notaremos distintos efectos: primero, que cada pico se vería ensanchado. Además, los valores en los que antes se anulaba $g(r)$ ahora pueden ser no-nulos debido a que algunos pares de partículas contribuyen a dar esas distancias. Para r grande el efecto se hace más evidente ya que se pierde el orden de largo alcance en el cristal. Continuando con lo anterior, para el caso de un fluido la función será nula hasta cierto r, lo cual es un efecto de la repulsión a corto alcance del potencial. A partir de tal distancia $g(r)$ será no-nula para los demas valores, con anchos picos que se deben a que el líquido es denso y por ello las distancias con los primeros vecinos seran mas o menos las mismas pero estas tendran mayor libertad de variar en el tiempo.

En la region de coexistencia se observaría un comportamiento intermedio.

Chapter 4

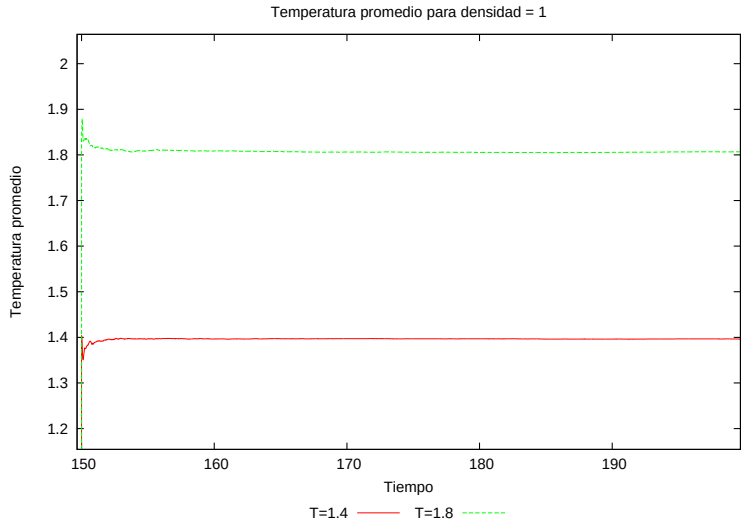
Resultados

4.1 Parámetros

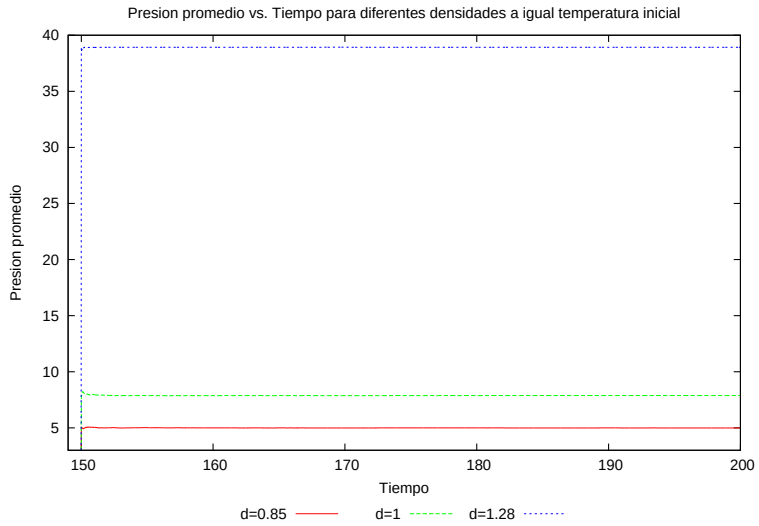
Se llevaron a cabo simulaciones calculando las mismas cantidades, mencionadas arriba. En total, iniciamos simulaciones con N_d densidades distintas para cada temperatura media diferente, de las cuales se tomaron N_T. Entonces, en total, fueron $N_d \times N_T$ simulaciones distintas. Para cada una de ellas tuvimos: Un sistema de 864 partículas dispuestas inicialmente en un arreglo FCC, con velocidades iniciales aleatorias con distribución gaussiana que correspondieran a la temperatura deseada para el sistema completo. 20000 pasos totales de los cuales fueron 15000 de termalización y 5000 de evolución. Se reescalearon las velocidades cada 20 pasos. El paso de tiempo fue siempre $\Delta t = 0.01$, el radio de corte $r_{corte} = 2.5$

Luego de termalizar procedemos a calcular promedios. Primeramente veamos el aspecto que tiene un gráfico de temperatura o presión medias para densidades y temperaturas iniciales intermedias.

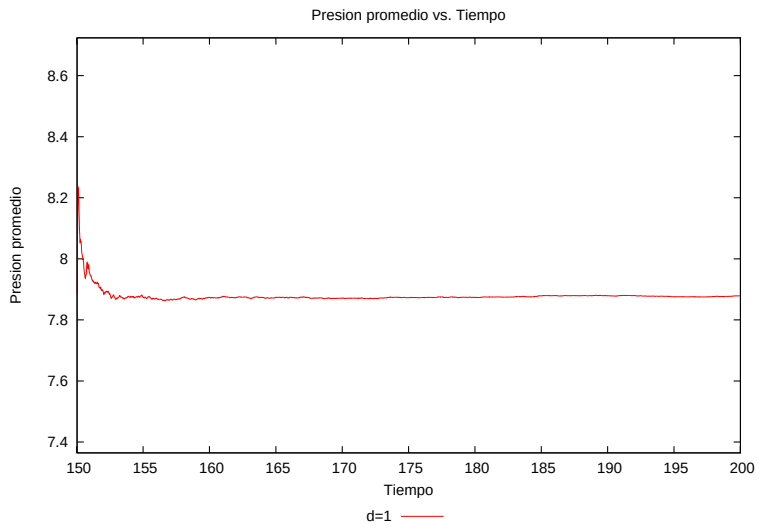
Por ejemplo, para densidad unidad, la evolución de la temperatura promedio en el tiempo posterior al de termalización tiene la siguiente forma:



Graficamos así también la presión promedio para una misma temperatura y diferentes densidades.

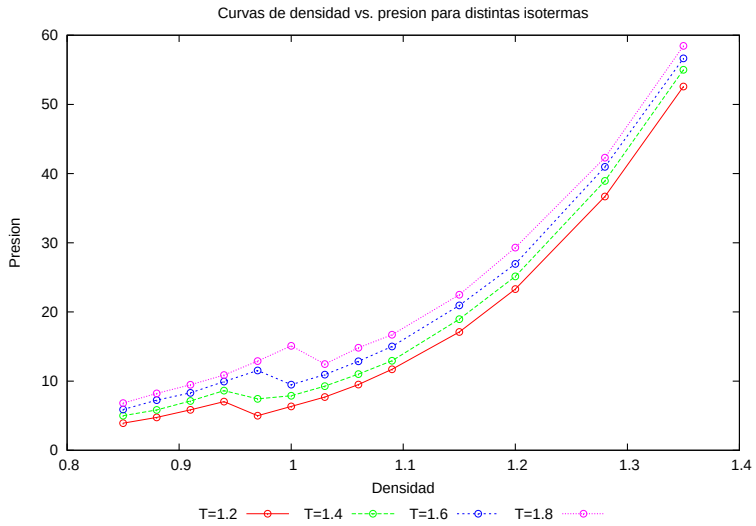


O podemos ver de cerca una sola



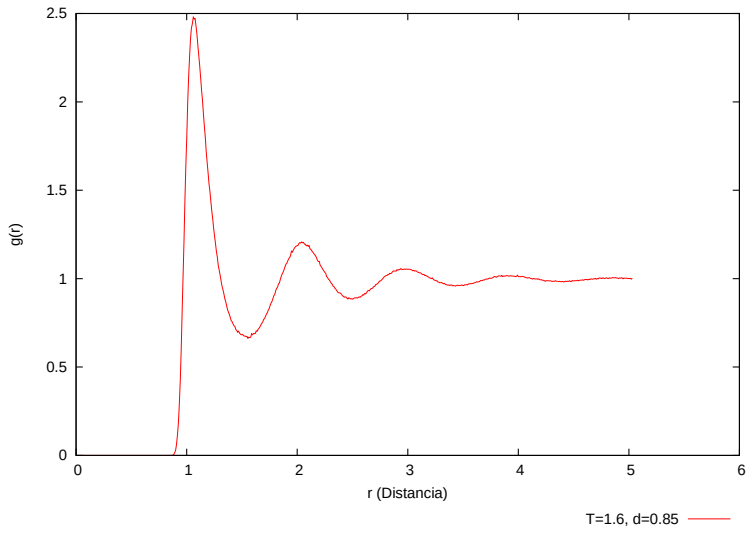
Se puede ver que, a pesar de las fluctuaciones, los valores se estabilizan en promedio. Entonces tenemos que cada sistema simulado corresponde a un punto en el plano ρ-T.

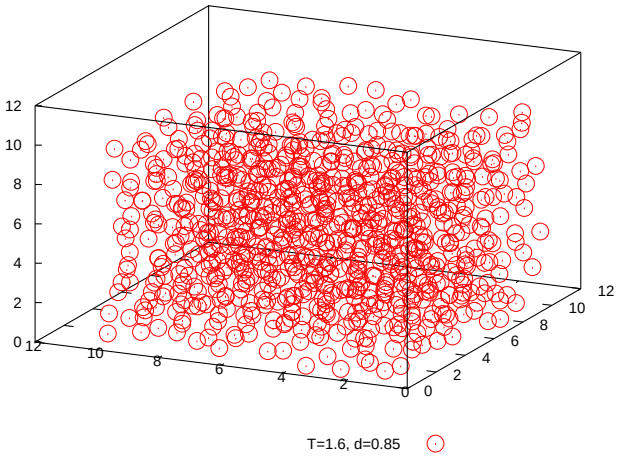
Finalmente podemos obtener curvas de ρ vs. P para distintas isotermas.



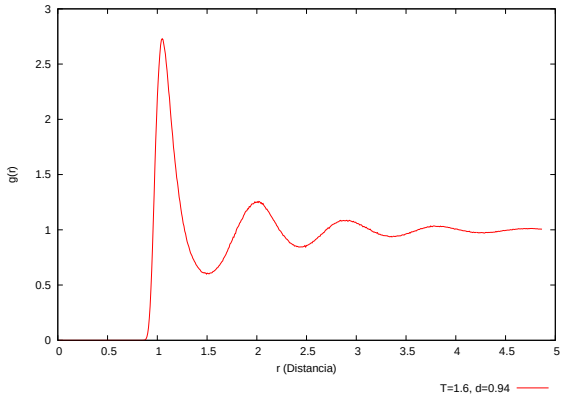
Ya con esto puede verse que existen ciertos puntos en los que parece haber una transición de fase. Para tener una mejor idea de lo que ocurre veamos antes, para una temperatura intermedia (entre las consideradas) el aspecto que tiene la FDR para densidades distintas. A estos graficos los contrastamos con una instantánea de las posiciones de las partículas la final de la simulación.

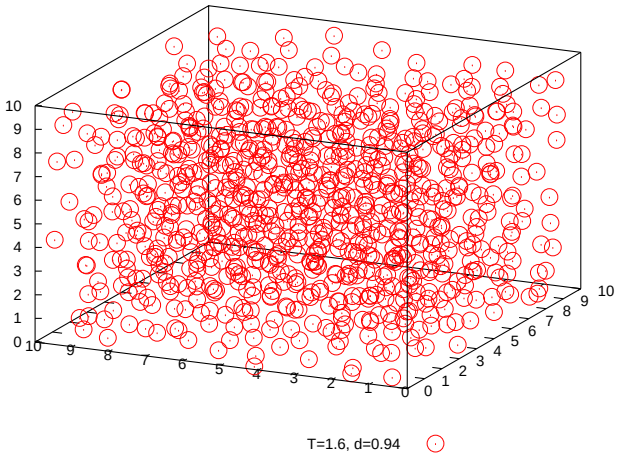
Tomamos $T = 1.6$ y empezamos con la densidad más baja de las simuladas:

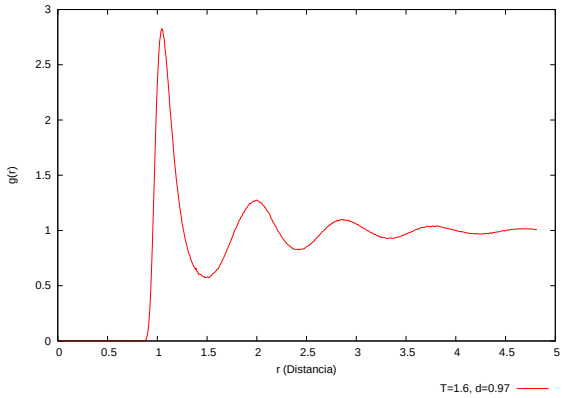


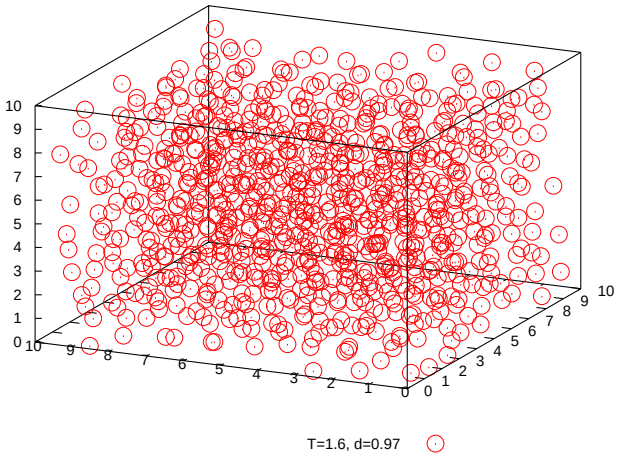


Ahora veremos qué ocurre cerca de las densidades en las que la relación $P(\rho)$ parece dejar de ser monótonamente creciente.

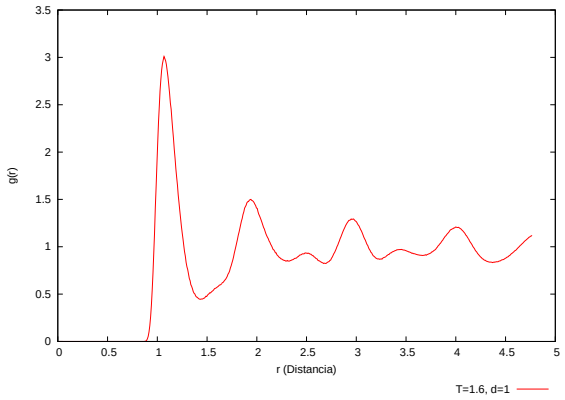


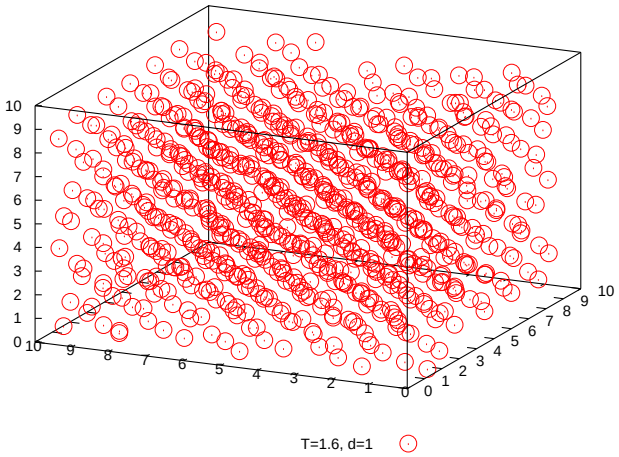

T=1.6, d=0.94

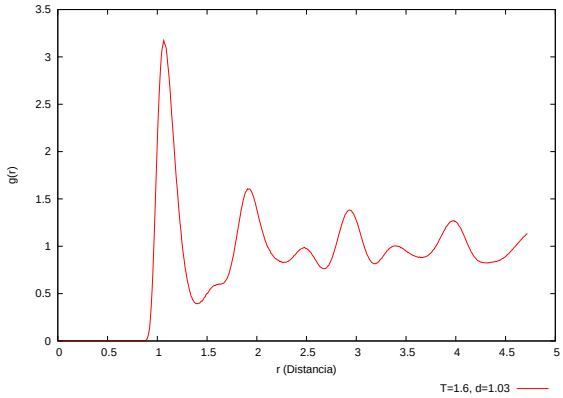

g(r)
r (Distancia)
T=1.6, d=0.97

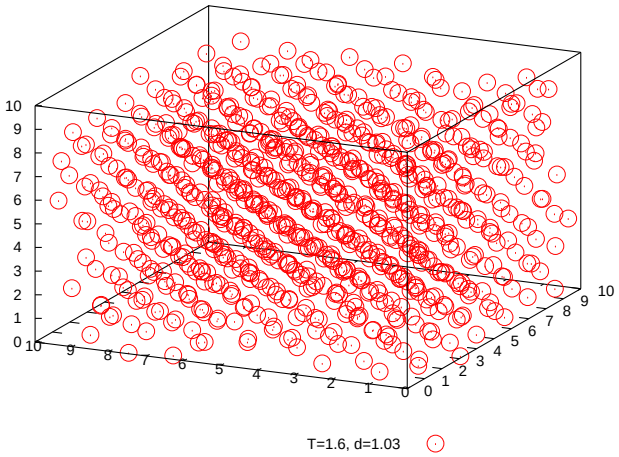


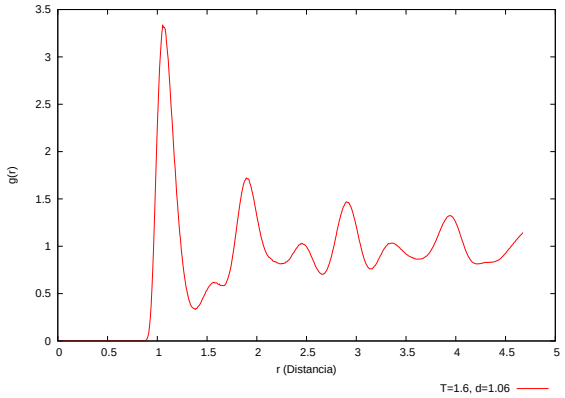
Desde aquí empieza a notarse ya una transición, ya que la forma de la FDR no corresponde precisamente a la de un líquido. Ciertamente, observando las posiciones de las partículas, puede verse que esto ocurre.

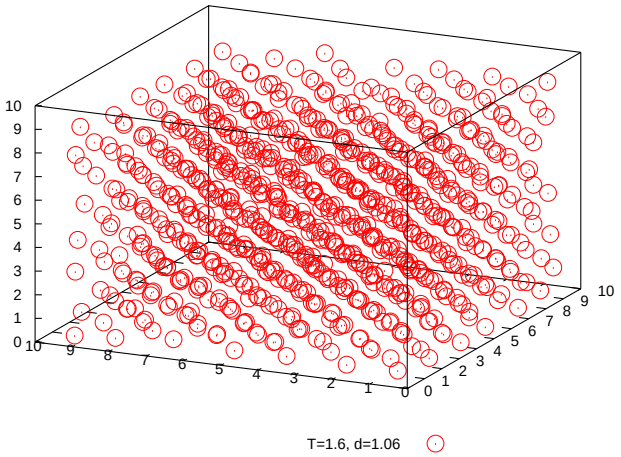


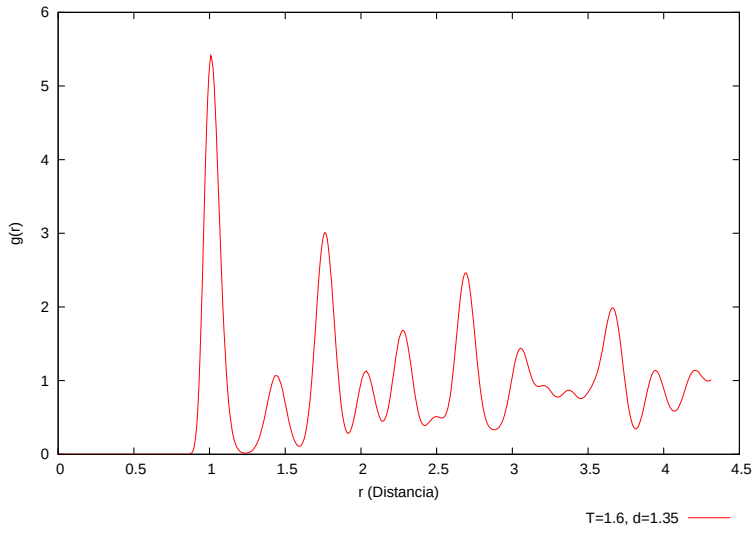

T=1.6, d=1

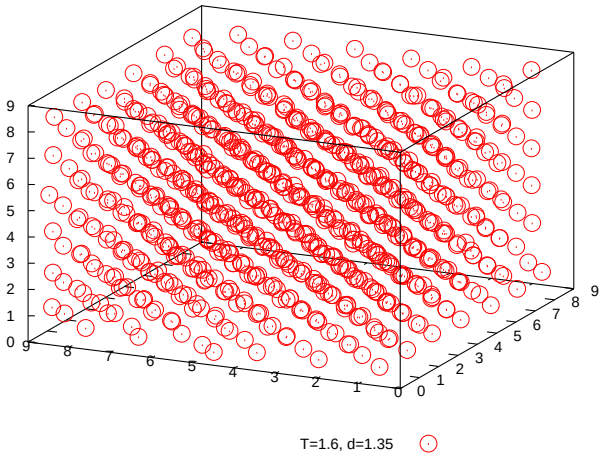

g(r)
r (Distancia)
T=1.6, d=1.03


T=1.6, d=1.03

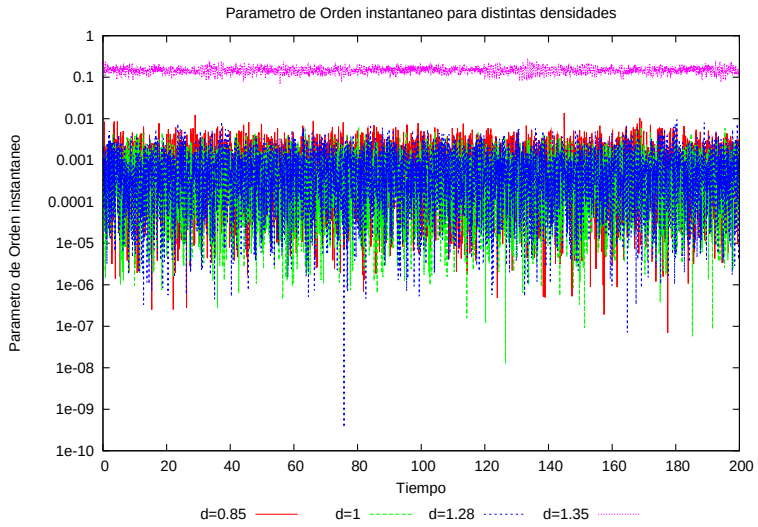

g(r)
r (Distancia)
T=1.6, d=1.06


T=1.6, d=1.06

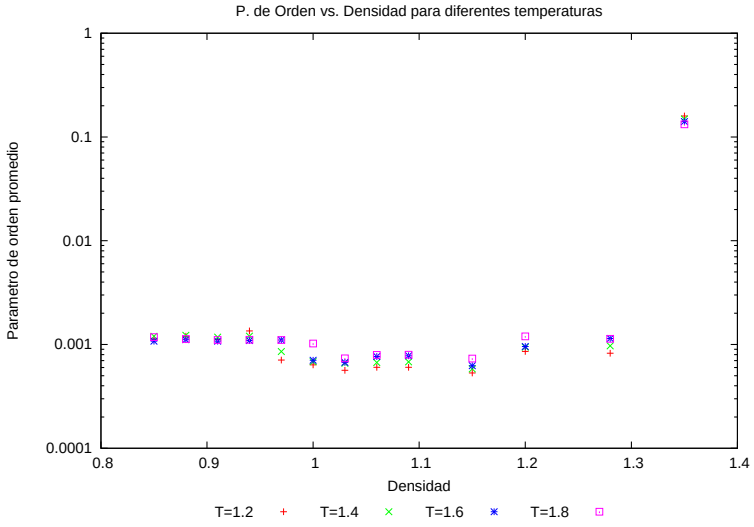

g(r)
r (Distancia)
T=1.6, d=1.35



Resulta útil también graficar el parametro de orden. A lo largo de cada corrida, los valores instantáneos se ven de la siguiente forma.



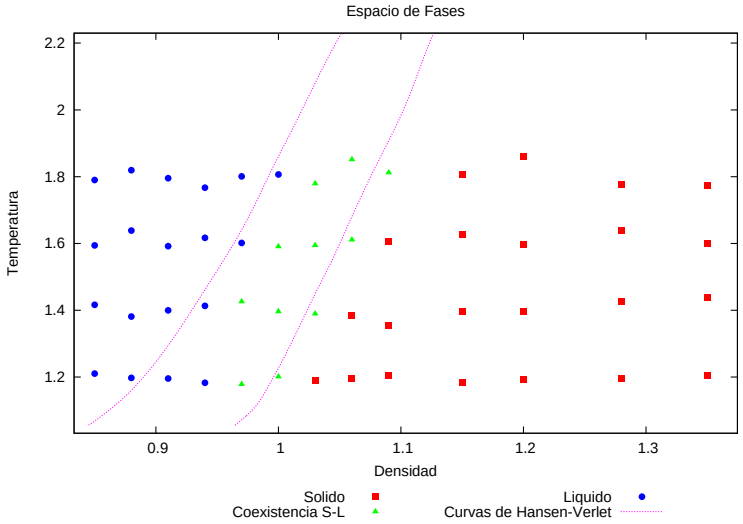
Podemos graficar el valor medio de estos luego de la termalización (aunque realmente parece no hacer mucha diferencia).


P. de Orden vs. Densidad para diferentes temperaturas
Parametro de orden promedio
1
0.1
0.01
0.001
0.0001
0.8
0.9
1
1.1
1.2
1.3
1.4
Densidad
T=1.2
T=1.4
T=1.6
T=1.8

Chapter 5

Análisis de resultados. Conclusiones

Combinando los datos mostrados anteriormente es posible determinar de que fase se trata cada sistema simulado.



La curva continua que aparece es la que fue obtenida tras las simulaciones efectuadas utilizando tecnicas montecarlo por Hansen y Verlet.

Parece existir acuerdo con los resultados que obtuvieron ellos, para un sistema similar pero efectuando importantes correcciones en las zonas de coexistencia.

Los resultados obtenidos parecen, en comparación, levemente desplazados hacia la derecha. Esto puede deberse a que no se aplicó ningún tipo de corrección a los efectos del radio de corte del potencial.

Chapter 6

Código fuente

El programa utilizado para las simulaciones fue escrito en C y se divide en dos archivos. *simlib.h* que contiene funciones de utilidad y *molecular.c* que contiene el código principal.

6.1 simlib.h

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

#define KI    0
#define VI    1
#define TI    2
#define PI    3
#define KP    4
#define VP    5
#define TP    6
#define PP    7
#define G     8
#define D     9
#define MODO 10
#define PRMS 11

#define MD    1
#define MC    2

#define X 0
#define Y 1
#define Z 2

#define ERRMSG_DT "Reducir dt."

typedef double vector[3];

//parsing de entrada, para determinar parametros de la simulacion
void parametros(
    int ac, char *av[],
```

```
  int *s, int *m, int *p, int *pt, int *ct,
  double *d, double *T,  double *dt, double *rc,
  double *dr) {
int i=1, inicio=0, lugar, k;
int ssn=0, sst=0, ssd=0, ssp=0, ssdt=0, ssct=0, sspt=0, ssrc=0, ssdr=0;
char st[30];

while(i<ac) {
  if(!inicio) {
    if(av[i][0]!='-') {
      i++;
    } else {
      lugar=i;
      inicio=1;
      if(lugar==1) {
        fprintf(stderr, "Argumentos...?\n");
        fprintf(stderr, "-n, -t, -d, -p, -dt, -ct, -pt, -rc, (-mc/-md), -dr
            *\n");
        exit(EXIT_FAILURE);
      }
    }
  } else {
    //if(c==9) break;
    //temperatura
    if (!strcmp(av[i],"-t")) {
      if(i<ac-1&&av[i+1][0]!='-') {
        *T=atof(av[++i]);
        sst=1;
      } else {
        printf("Especifique el valor para %s.\n",av[i]);
        exit(1);
      }
    // densidad
    } else if (!strcmp(av[i],"-d")) {
      if(i<ac-1&&av[i+1][0]!='-') {
        *d=atof(av[++i]);
        ssd=1;
      } else {
        printf("Especifique el valor para %s.\n",av[i]);
        exit(1);
      }
    // pasos
    } else if (!strcmp(av[i],"-p")) {
      if(i<ac-1&&av[i+1][0]!='-') {
        *p=atoi(av[++i]);
        ssp=1;
      } else {
        printf("Especifique el valor para %s.\n",av[i]);
        exit(1);
      }
    // dt
    } else if (!strcmp(av[i],"-dt")) {
      if(i<ac-1&&av[i+1][0]!='-') {
        *dt=atof(av[++i]);
```

```
    ssdt=1;
  } else {
    printf("Especifique el valor para %s.\n",av[i]);
    exit(1);
  }
// ct
}  else if (!strcmp(av[i],"-ct")) {
  if(i<ac-1&&av[i+1][0]!='-') {
    *ct=atoi(av[++i]);
    ssct=1;
  } else {
    printf("Especifique el valor para %s.\n",av[i]);
    exit(1);
  }
// pt
}  else if (!strcmp(av[i],"-pt")) {
  if(i<ac-1&&av[i+1][0]!='-') {
    *pt=atoi(av[++i]);
    sspt=1;
  } else {
    printf("Especifique el valor para %s.\n",av[i]);
    exit(1);
  }
// rc
}  else if (!strcmp(av[i],"-rc")) {
  if(i<ac-1&&av[i+1][0]!='-') {
    *rc=atof(av[++i]);
    ssrc=1;
  } else {
    printf("Especifique el valor para %s.\n",av[i]);
    exit(1);
  }
// M
}  else if (!strcmp(av[i],"-n")) {
  if(i<ac-1&&av[i+1][0]!='-') {
    *m=atoi(av[++i]);
    ssn=1;
  } else {
    printf("Especifique el valor para %s.\n",av[i]);
    exit(1);
  }
} else if (!strcmp(av[i],"-dr")) {
  if(i<ac-1&&av[i+1][0]!='-') {
    *dr=atof(av[++i]);
    ssdr=1;
  } else {
    printf("Especifique el valor para %s.\n",av[i]);
    exit(1);
  }
// modo (mc o md)
} else if(((!strcmp(av[i],"-mc"))||(!strcmp(av[i],"-md")))) {
  if(!strcmp(av[i],"-md")) {
    s[MODO]=MD;
  } else if(!strcmp(av[i],"-mc")) {
```

```
            s[MODO]=MC;;
          } else {
            fprintf(stderr, "Modo?\n");
            exit(1);
          }
        }
        i++;
      }
    }
   /*
    if( ( (s[MODO]==MD)&&(c!=9) )
       ||( (s[MODO]==MC)&&(c!=10) )
      ) {*/
    if(sst&&ssd&&ssn&&ssp&&ssdt&&ssct&&sspt&&ssrc) {
      if(s[MODO]==MD) {
        sprintf(st, "Dinamica Molecular");
      } else if(s[MODO]==MC&&ssdr) {
        sprintf(st, "Monte Carlo");
      }
      printf("##############################################\n");
      printf("# Simulando en modo %s\n", st);
      printf("# ========================================\n");
      printf("# (Imprimiendo:");
      for(k=1;k<lugar;k++) {
        printf(" %s", av[k]);
      }
      printf(")\n");

      printf("##############################################\n");
      printf("# N = %d\n# (%d particulas)\n#\n", *m, 4*(*m)*(*m)*(*m));
      printf("# Temperatura = %g\n", *T);
      printf("# Densidad = %g\n", *d);
      printf("# Pasos totales = %d\n", *p);
      printf("# Paso de tiempo (dt) = %g\n", *dt);
      printf("# Pasos de termalizacion = %d\n", *pt);
      printf("# Intervalo de termalizacion = %d\n", *ct);
      printf("# Radio de corte=%g\n", *rc);
      printf("##############################################\n");
    } else {
      fprintf(stderr, "Falta especificar algun parametro inicial!\n");
      fprintf(stderr, "-n (el num de particulas es 4*n*n*n), -t (temperatura), -
          d (densidad), -p (pasos totales), -ct (termalizar cada ct pasos
          mientras dura la termalizacion), -pt (cantidad de pasos de
          termalizacion), -dt (paso de tiempo), -rc, (-mc/-md) (montecarlo o
          dinamica molecular), -dr* (paso de movimiento, para modo mc)\n");
      exit(1);
    }
}

//del parsing fija ciertos valores, por ej nro de particulas, lado de caja (v
    ^1/3), etc
void iniciar_ctes(int *m, int *n, double *d, double *lcel, double *lcaj) {
  (*n) = 4*((*m)*(*m)*(*m));
  (*lcel) = pow(4/(*d),1/3.);
```

```
  (*lcaj) = (*lcel)*(*m);
}

// mas parsing, esto para determinar que es lo que pide el usuario que se
    calcule
// por ej ki (energia cinetica instantanea), kp (e. cinetica promedio),
// g (f. de dist radial), d (relacion de difusion), etc
void interruptores(int ac, char *av[], int *s) {
  int i=1,j,k=0;
  char *cads[] = {
    "ki", "vi", "ti", "pi",
    "kp", "vp", "tp", "pp",
     "g", "d"
  };

  for(j=0;j<10;j++) s[j]=0;

  while(i<ac&&av[i][0]!='-') {
    for(j=0;j<10;j++) {
      if(!strcmp(av[i],cads[j])) {
        s[j]=1;
        k++;
      }
    }
    if(k!=i) {
      fprintf(stderr, "No se reconoce la opcion ingresada.\nLas opciones
          disponibles son: ");
      for(j=0;j<10;j++) fprintf(stderr, "%s ", cads[j]);
      fprintf(stderr,"\n");
      exit(EXIT_FAILURE);
    }
    i++;
  }
  if(k==0) {
    fprintf(stderr, "Debe escoger al menos una opcion: ");
    for(j=0;j<10;j++) fprintf(stderr, "%s ", cads[j]);
    fprintf(stderr,"\n");
    exit(EXIT_FAILURE);
  }
}

//determina dependencias (por ej, aunque solo desee kp es necesario calcular
    ki)
void dependencias(int *vs, int *s) {
  int k;

  for(k=0;k<12;k++) {
    if(k==MODO) continue;
    else s[k]=0;
  }

  if(vs[KP]||vs[VP]||vs[TP]||vs[PP]||vs[G]) s[PRMS]=1;
  if(s[PRMS]) {
    if(vs[PP]) {
```

```
      s[PP]=1;
      s[PI]=1;
      s[KP]=1;
      s[PP]=1;
    }
    if(vs[TP]) {
      s[TP]=1;
      s[TI]=1;
      if(s[MODO]==MD) {
        s[KP]=1;
        s[KI]=1;
      }
    }
    if(vs[KP]) {
      s[KP]=1;
      s[KI]=1;
    }
  }
  for(k=0;k<KP;k++) {
    if(vs[k]) s[k]=1;
  }
  for(k=8;k<MODO;k++) {
    if(vs[k]) s[k]=1;
  }

  if((s[MODO]==MC)&&(vs[KI]||vs[KP]||vs[G]||vs[TI]||vs[TP])) {
    fprintf(stderr, "Se escogio una opcion no compatible con el modo MC.\n");
    exit(EXIT_FAILURE);
  }
}

// constantes para la f. de dist. radial
void iniciar_ctes_g(int ac, char *av[], int *b) {
  int i=1, c=0;

  while(i<ac) {
    if (!strcmp(av[i],"-b")) {
      if(i<ac-1&&av[i+1][0]!='-') {
        *b=atoi(av[++i]);
        c++;
      } else {
        printf("Especifique el valor para %s.\n",av[i]);
        exit(1);
      }
    break;
    }
    i++;
  }

  if(c!=1) {
    fprintf(stderr,"Especificar -b\n");
    exit(1);
  }
}
```

```
// constantes para la curva de difusion
void iniciar_ctes_d(int ac, char *av[], int *it0, int *t0max, int *tmax, int *
    cadad) {
  int i=1, c=0;

  while(i<ac) {
    // it0
    if (!strcmp(av[i],"-t0")) {
      if(i<ac-1&&av[i+1][0]!='-') {
        *it0=atoi(av[++i]);
        c++;
      } else {
        printf("Especifique el valor para %s.\n",av[i]);
        exit(1);
      }
    // t0max
    } else if (!strcmp(av[i],"-t0m")) {
      if(i<ac-1&&av[i+1][0]!='-') {
        *t0max=atoi(av[++i]);
        c++;
      } else {
        printf("Especifique el valor para %s.\n",av[i]);
        exit(1);
      }
    // tmax
    } else if (!strcmp(av[i],"-tm")) {
      if(i<ac-1&&av[i+1][0]!='-') {
        *tmax=atoi(av[++i]);
        c++;
      } else {
        printf("Especifique el valor para %s.\n",av[i]);
        exit(1);
      }
    // cadad
    }  else if (!strcmp(av[i],"-cd")) {
      if(i<ac-1&&av[i+1][0]!='-') {
        *cadad=atoi(av[++i]);
        c++;
      } else {
        printf("Especifique el valor para %s.\n",av[i]);
        exit(1);
      }
    }
    i++;
  }

  if(c!=4) {
    fprintf(stderr, "Falta especificar alguno de los siguientes: -t0 (cada
        cuantos muestreos tengo un nuevo origen de tiempo), -t0m (cantidad
        maxima de origenes de tiempo), -tm (cantidad de tiempos o puntos de la
        curva), -cd (cada cuantos pasos de evolucion muestreo)\n");
    exit(1);
  }
```

```
}

// duh
void iniciar_valores_inst(int *s, double *ki, double *vi, double *w) {
  if(s[KI]) *ki=0;
  if(s[VI]) *vi=0;
  if(s[PI]) *w=0;
}

#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)

// ran1, de Numerical Recipes in C
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;
}

#undef IA
#undef IM
#undef AM
#undef IQ
#undef IR
#undef NTAB
```

```
#undef NDIV
#undef EPS
#undef RNMX

void serror(char *s) {
  fprintf(stderr,"%s\n",s);
  exit(EXIT_FAILURE);
}

// gasdev, de Numerical Recipes in C
double gasdev(long *idum) {
  static int iset=0;
  static double gset;
  double fac,rsq,v1,v2;

  if(*idum<0) iset=0;
  if(iset==0) {
    do {
      v1=2.0*ran1(idum)-1.0;
      v2=2.0*ran1(idum)-1.0;
      rsq=v1*v1+v2*v2;
    } while (rsq >= 1.0 || rsq == 0.0);
    fac=sqrt(-2.0*log(rsq)/rsq);
    gset=v1*fac;
    iset=1;
    return v2*fac;
  } else {
    iset=0;
    return gset;
  }
}

void iniciar_posiciones(int N, vector *p, double l) {
  int m, i, j, k, q;
  vector pos;
  vector b[4] = {
    {0.0,0.0,0.0},
    {0.5,0.5,0.0},
    {0.5,0.0,0.5},
    {0.0,0.5,0.5}
  };

  for(i=0;i<N;i++)
    for(j=0;j<N;j++)
      for(k=0;k<N;k++) {
        pos[X]=i; pos[Y]=j; pos[Z]=k;
        for(q=0;q<4;q++,p++)
          for(m=0;m<3;m++)
            (*p)[m]=(pos[m]+b[q][m])*l;
      }
}

void imprimir_posiciones(int N, vector *p) {
  int i,k;
```

```
  printf("\n\n");

  for(i=0;i<N;i++,p++) {
    printf("%d\t",i);
    for(k=0;k<3;k++) printf("%e ",(*p)[k]);
    printf("\n");
  }
}

void iniciar_posiciones_random(int N, vector *p, double l) {
  int i,k;
  long int s=-13;

  for(i=0;i<N;i++,p++) {
    for(k=0;k<3;k++) (*p)[k]=ran1(&s)*l;
  }
}

// esto fue util para los tiempos de debugging, no lo uso pero ahi quedo
void cargar_posiciones(int N, vector *p, double l) {
  FILE *f;
  char fn[100], c;
  int i,n=0;

  printf("Archivo de posiciones: ('.' para posiciones.dat)\n");
  scanf("%s",fn);
  if(strcmp(fn,".")==0) sprintf(fn,"posiciones.dat");

  if((f=fopen(fn,"r"))==NULL) serror("No se puede abrir el archivo.");

  do {
    c=getc(f);
    if(c=='\n') n++;
  } while(c!=EOF);
  if(n!=N) serror("Numero de lineas y de particulas no concuerdan.");
  rewind(f);

  for(i=0;i<n;i++,p++) {
    fscanf(f,"%lf\t%lf\t%lf\n",&((*p)[X]),&((*p)[Y]),&((*p)[Z]));
  }
}

void iniciar_velocidades(int N, vector *v, double T) {
  int i, k;
  static long int s=0;
  double rt=sqrt(T);

  if(!s) {
    srand(time(NULL));
    s=-(long)rand();
  }

  for(i=0;i<N;i++,v++) {
```

```
    for(k=0;k<3;k++) (*v)[k] = gasdev(&s)*rt;
  }
}

void sumar_fuerza(
    double *p_i, double *p_j,
    double *f_i, double *f_j,
    double *w_inst, double *energia_p_inst, double l,
    int s_fuerza, int s_presion_inst, int s_ep_inst
    ) {
  static double rc2 = R_CORTE*R_CORTE;
  int k;
  double r2=0, rm2, rm6, f;
  vector d;

  for(k=0;k<3;k++,p_i++,p_j++) {
    d[k]=*p_i-*p_j;

    if(fabs(d[k])>0.5*l)
      d[k]-=(d[k]>=0?l:-l);
  }

  for(k=0;k<3;k++) r2+=d[k]*d[k];
  if(r2>=rc2) return;
  if(r2==0) r2=1e-6;

  rm2=1/r2;
  rm6=rm2*rm2*rm2;

  f=48*(rm6*(rm6-0.5));

  if(s_fuerza) {
    if(s_presion_inst) *w_inst+=f;
    if(s_ep_inst) *energia_p_inst+=4*rm6*(rm6-1);
  }

  f*=rm2;

  for(k=0;k<3;k++,f_i++,f_j++) {
    (*f_i) += f*d[k];
    (*f_j) -= f*d[k];
  }
}

int verificar_condcontorno(int N, vector *p, double l) {
  int i, k;

  for(i=0;i<N;i++,p++) {
    for(k=0;k<3;k++) {
      if((*p)[k]>l) (*p)[k]-=l;
      else if ((*p)[k]<0) (*p)[k]+=l;

      if((*p)[k]>l||(*p)[k]<0) return 1;
    }
```

```
  }

  return 0;
}

double calcular_energia_k(int N, vector *v) {
  int i,k;
  double s=0;

  for(i=0;i<N;i++,v++)
    for(k=0;k<3;k++) s+=(*v)[k]*(*v)[k];

  return 0.5*s;
}

void termalizar(int N, vector *v, double T, double T_inst) {
  int i,k;
  double a=sqrt(T/T_inst);

  for(i=0;i<N;i++,v++)
    for(k=0;k<3;k++) (*v)[k]*=a;
}

double min_dist2(double *p_i, double *p_j, double l) {
  int k;
  double s=0;
  vector d;

  for(k=0;k<3;k++,p_i++,p_j++) {
    d[k]=*p_i-*p_j;

    if(fabs(d[k])>0.5*l)
      d[k]-=(d[k]>=0?l:-l);

    s+=d[k]*d[k];
  }

  return s;
}

double dist(double *p_i, double *p_j) {
  int k;
  double s=0;

  for(k=0;k<3;k++,p_i++,p_j++)
    s+=(*p_i-*p_j)*(*p_i-*p_j);

  return sqrt(s);
}

double pasitoMC(int N, int n, vector *p, double l, double T, double *dr,
   double *a) {
  int i, k;
  static int acep=0, reje=0, setdr=0;
```

```
static double rcorte2=R_CORTE*R_CORTE, ds=0;
double r2, rm6;
vector *pn=&(p[n]),*p0=p;
vector pp;
double u=0, u0=0, du=0;
static long int s=0;

if(!s) srand(time(NULL));
s=-(long)rand();

if(!setdr) {
  ds=*dr;
  setdr=1;
}
else if(*dr!=ds) {
  acep=0;
  reje=0;
  ds=*dr;
}

for(i=0;i<N;i++,p++) {
  if(i!=n) {
    r2=min_dist2(*p,*pn,l);

    if(r2<rcorte2) {
      rm6=pow(r2,-3);
      u0+=4*rm6*(rm6-1);
    }
  }
}

for(k=0;k<3;k++) {
  pp[k]=(*pn)[k]+ds*(ran1(&s)-0.5);
  if(pp[k]>l) pp[k]-=l;
  else if (pp[k]<0) pp[k]+=l;
}

for(i=0,p=p0;i<N;i++,p++) {
  if(i!=n) {
    r2=min_dist2(*p,pp,l);
    if(r2<rcorte2) {
      rm6=pow(r2,-3);
      u+=4*rm6*(rm6-1);
    }
  }
}

du=u-u0;
if(ran1(&s)<exp(-du/T)) {
  for(k=0;k<3;k++) (*pn)[k]=pp[k];
  acep++;
} else {
  du=0;
  reje++;
```

```
  }

  *a=((double)acep)/((double)(acep+reje));

  return du;
}

double calcular_energia_p(int N, vector *p, double l) {
  double r2, rm6;
  double v=0;
  static double rc2=R_CORTE*R_CORTE;
  int i, j;

  for(i=0;i<N;i++) {
    for(j=i+1;j<N;j++) {
      r2=min_dist2(p[i],p[j],l);
      if(r2>=rc2) continue;
      rm6=pow(r2,-3);
      v+=rm6*(rm6-1);
    }
  }
  v*=4;

  return v;
}

double calcular_presion(int N, vector *p, double l, double T, double d) {
  int i, j;
  static double rc2=R_CORTE*R_CORTE;
  double r2, rm6, w=0;

  for(i=0;i<N;i++) {
    for(j=i+1;j<N;j++) {
      r2=min_dist2(p[i],p[j],l);
      if(r2<rc2) {
        rm6=pow(r2,-3);
        w+=rm6*(rm6-0.5);
      }
    }
  }
  w*=48;

  return d*(T+(w/(3*N)));
}

double p_orden(int N, vector *p, double l) {
  double c=2*3.141593*l*pow(((double)N)/4.,-1/3.);
  vector k;
  double ssen=0,scos=0,dp;
  double un=pow(N,-2);
  int i,j;

  k[0]=-c; k[1]=c; k[2]=-c;
```

```
  for(i=0;i<N;i++) {
    dp=0;
    //producto punto
    for(j=0;j<3;j++) dp+=k[j]*p[i][j];
    ssen+=sin(dp); scos+=cos(dp);
  }

  return (ssen*ssen+scos*scos)*un;
}

/*
void visualizar(vector *p) {
  int i,k;
  FILE *f = fopen("datapos","w");
  FILE *pipe = popen("gnuplot -p","w");

  for(i=0;i<NUM_PARTICULAS;i++,p++) {
    for(k=0;k<3;k++) {
      fprintf(f,"%e\t",(*p)[k]);
    }
    printf("\n");
  }
  fclose(f);

  fprintf(pipe, "splot \"datapos\" pt 7\n");
  fclose(pipe);
}*/

void fdistradial(int N, vector *p, double l, int b) {
  static double *distribucion;
  static long int n=0;
  static int gset=0, bins=0;
  static double hl, dr;
  double v, nn;
  int i,j;

  if(b>0) {
    if(!gset) {
      hl=l*0.5;
      bins=b;
      distribucion=(double*)malloc(sizeof(double)*(bins+1));
      if(distribucion==NULL) serror("Memoria insuficiente. Reduzca numero de
          bins.");

      gset=1;
      dr=hl/((double)bins);
    }
    else {
      n++;
      for(i=0;i<N-1;i++) {
        for(j=i+1;j<N;j++) {
          distribucion[(int)floor(sqrt(min_dist2(p[i],p[j],l))/dr)]+=2.;
        }
      }
```

```
    }
  }
  else {
    printf("\n\n");
    for(i=1;i<=bins;i++) {
      v=(pow(i+1,3)-pow(i,3))*pow(dr,3);
      nn=(4/3.)*3.141593*v*l;

      printf("%e\t%e\n",dr*(i+0.5),(distribucion[i-1])/(n*nn*N));
    }
    // no entiendo por que me da segfault con estoooo!!!! si saco esta todo
       bien, supongo que no hace mucho dano al mundo
    //free(distribucion);
  }
}

/* parametros:
   it0 (indica cada cuantos muestreos tengo un nuevo origen de tiempos),
   tmax, indica la cantidad maxima de deltat (dentro de la rutina),
   t0max, (cadad,timestep (al final))
*/
// quiero pasarle la posicion de las particulas (sin verificar condiciones de
   contorno, a partir de que finaliza la termalizacion)
// para el final necesito pasarle 'cadad'
void difusion(int N, vector *x, int it0, int t0max, int tmax, double dtime) {
  static int dset=0, ntel=0, t0=0, *time0;
  static double *ntime, *r2t;
  static vector **x0;
  int i, k, l, tt0, delt;

  // inicializar variables para coef de difusion...
  // ntel=0,r2t[0:tmax-1]=0,ntime[0:tmax-1],dtime=dt*cadad
  //  (se muestrea cada cadad pasos)
  //
  if(x!=NULL) {
    if(!dset) {
      // alocar memoria para vectores double 0:tmax-1; ntime, vacf y r2t
      ntime=(double *)malloc(sizeof(double)*(tmax+1));
      r2t=(double *)malloc(sizeof(double)*(tmax+1));
      time0=(int *)malloc(sizeof(int)*(t0max+1));
      // ahora para las posiciones (y velocidades) hasta cierto tiempo
         limitado
      x0=malloc(sizeof(vector*)*(N+1));
      for(i=0;i<N;i++) {
        x0[i]=malloc(sizeof(vector)*(t0max+1));
        if(x0[i]==NULL) serror("Error al alocar memoria. Disminuir t0max");
      }

      // VERIFICAR ALOCAMIENTO CORRECTO!!!!
      if(ntime==NULL||r2t==NULL)
        serror("No se ha podido alocar memoria. Reducir tmax o algun otro
           parametro.");
      dset=1;
    }
```

```
    // muestreamos cada 'cadad' pasos luego de la termalizacion
    // (en esta parte no existe referencia a 'cadad')
    else /**/ {
      // nuevo origen de tiempos, cada cadad*it0 pasos
      if(ntel%it0==0) {
        tt0=t0%t0max;
        time0[tt0]=ntel;
        // posiciones
        for(i=0;i<N;i++) {
          for(k=0;k<3;k++) {
            x0[i][tt0][k]=x[i][k];
          }
        }
        t0++;
      }
      // se actualizan, hasta l<min(t0,t0max)
      for(l=0;l<((t0<t0max)?t0:t0max);l++) {
        delt=ntel-time0[l];
        if(delt<tmax) {
          ntime[delt]++;
          for(i=0;i<N;i++) {
            r2t[delt]+=dist(x[i],x0[i][l]);
          }
        }
      }
      ntel++;
    }
  }
  else {
    printf("\n\n");
    for(i=0;i<tmax;i++) {
      // t0max<-cadad (p/ ahorrar unos cuantos bits)
      r2t[i]/=t0max*ntime[i];
      printf("%e\t%e\n",dtime*(i+0.5), r2t[i]);
    }
    free(ntime);
    free(r2t);
    free(time0);
    for(i=0;i<N;i++) {
      free(x0[i]);
    }
    free(x0);
  }
}
```

6.2 molecular.c

```
// el codigo bien escrito habla por si solo, sin necesidad de muchos
    comentarios... uds diganme

#include "simlib.h"

int main(int argc, char *argv[]) {
  int i, j, k, p;
```

```
int swv[10], sw[12], cv=0, cvi=0;
double **pv, *pa[8];
int m=0, n;
double densidad, temperatura, lado_celda, lado_caja;
double dt, rcorte;

double param_orden,suma_s=0,veces_s=0;

double dr, acep;

int p_termalizacion, pasos, cada;
int bins;
int cadad=10, it0=12, t0max=20, tmax=200;

int s_fuerza, s_termalizacion, s_c_difusion=0;

double t, pev;
double energia_k_inst,   energia_k_prom,   energia_k_suma,
       energia_p_inst,   energia_p_prom,   energia_p_suma,
       w_inst,           w_prom,           w_suma,
       presion_inst,     presion_prom,     presion_suma,
       temperatura_inst, temperatura_prom;
vector *posicion, *velocidad, *fuerza;

vector *x;
double aux;

// pa es un array de punteros a las distintas variables
// (usado para ordenarlas con un indice)
pa[KI]=&energia_k_inst;
pa[VI]=&energia_p_inst;
pa[TI]=&temperatura_inst;
pa[PI]=&presion_inst;
pa[KP]=&energia_k_prom;
pa[VP]=&energia_p_prom;
pa[TP]=&temperatura_prom;
pa[PP]=&presion_prom;

// lee los argumentos de entrada y guarda parametros especificados
parametros(argc, argv, sw, &m,
    &pasos, &p_termalizacion, &cada,
    &densidad, &temperatura, &dt, &rcorte, &dr);

iniciar_ctes(&m, &n, &densidad, &lado_celda, &lado_caja);

// pedimos memoria para los vectores posicion de cada particula
// (asi como velocidad y fuerza, si se selecciona el modo MD)
posicion=malloc(sizeof(vector)*(n+1));
if(sw[MODO]==MD) {
  velocidad=malloc(sizeof(vector)*(n+1));
  fuerza=malloc(sizeof(vector)*(n+1));
}

// lee la linea de entrada para saber que cosas calcular
```

```
interruptores(argc, argv, swv);

for(i=0;i<G;i++) {
  cv+=swv[i];
  if(i<KP) cvi+=swv[i];
}

// array de punteros a las variables seleccionadas
pv=malloc(sizeof(double*)*cv);
for(i=0,j=0;i<G;i++) {
  if(swv[i]) {
    pv[j]=pa[i];
    j++;
  }
}

// arregla dependencias (por ej., aun si solo pedimos imprimir promedios,
// es de todas formas necesario calcular valores instantaneos)
dependencias(swv,sw);

// d por curva de difusion, g por funcion de distribucion radial
if(sw[G]) iniciar_ctes_g(argc, argv, &bins);
if(sw[D]) {
  iniciar_ctes_d(argc, argv, &it0, &t0max, &tmax, &cadad);
  if((t0max*it0)<tmax) {
    serror("t0max*it0<tmax");
  }
  if(cadad*it0*t0max>pasos-p_termalizacion) {
    serror("cadad*it0*t0max>pasos-p_termalizacion");
  }
}

iniciar_posiciones(m, posicion, lado_celda);

if(sw[MODO]==MD) iniciar_velocidades(n, velocidad, temperatura);
if(sw[MODO]==MC) energia_p_inst=calcular_energia_p(n, posicion, lado_caja);

// iniciamos variables para promedios
if(sw[PRMS]) {
  if(sw[PP]) {
    presion_suma=0;
    w_suma=0;
    w_prom=0;
  }
  if(sw[KP]) {
    energia_k_suma=0;
    energia_k_prom=0;
  }
  if(sw[VP]) {
    energia_p_suma=0;
    energia_p_prom=0;
  }
  if(sw[G]) {
    fdistradial(n,posicion,lado_caja,bins);
```

```
  }
  if(sw[D]) {
    s_c_difusion=0;
  }
}

// bucle principal
////////////////////////////////////////////////////
for(p=1,pev=0,t=0,s_termalizacion=1;
    p<=pasos;p++,t+=dt,pev+=!s_termalizacion) {
  // inicializacion (cero)
  if(sw[MODO]==MD) iniciar_valores_inst(sw,&energia_k_inst,&energia_p_inst,
      &w_inst);
  else if(sw[MODO]==MC) {

  }

  if(sw[MODO]==MC) {
    for(i=0;i<n;i++) {
      energia_p_inst+=pasitoMC(n,i,posicion,lado_caja,temperatura,&dr,&acep)
          ;
    }
  }

  if(sw[MODO]==MD) {
    for(i=0;i<n;i++) {
      for(k=0;k<3;k++) fuerza[i][k]=0;
    }

    // calcula fuerzas
    /* ademas de Ep y W en caso de requerir el calculo de
       Ep y P instantaneos respectivamente */
    s_fuerza=1;
    for(i=0;i<n;i++) {
      for(j=i+1;j<n;j++) {
        sumar_fuerza(posicion[i] , posicion[j],
                     fuerza[i], fuerza[j],
                     &w_inst, &energia_p_inst,
                     lado_caja,s_fuerza,
                     sw[PI],sw[VI]);

      }
    }

    // calcula posicion en t+h
    for(i=0;i<n;i++) {
      for(k=0;k<3;k++) {
        if(s_c_difusion) {
          aux=dt*(velocidad[i][k]+0.5*dt*fuerza[i][k]);
          posicion[i][k]+=aux;
          x[i][k]+=aux;
        }
        else {
          posicion[i][k]+=dt*(velocidad[i][k]+
```

```
                        0.5*dt*fuerza[i][k]);
      }
    }
  }

  // suma fuerza en t+dt, para calcular aceleracion promedio
  s_fuerza=0;
  for(i=0;i<n;i++) {
    for(j=i+1;j<n;j++) {
      sumar_fuerza(posicion[i], posicion[j],
                   fuerza[i], fuerza[j],
                   NULL, NULL,
                   lado_caja,s_fuerza,
                   sw[PI],sw[VI]);
    }
  }

  // verifica cond de contorno y termina si dt es muy grande
  if(verificar_condcontorno(n,posicion,lado_caja)) {
    serror(ERRMSG_DT);
    return 1;
  }

  energia_k_inst = calcular_energia_k(n, velocidad);
  temperatura_inst = 2*energia_k_inst/(3*n);
  presion_inst = densidad*(temperatura_inst+w_inst/(3*n));

  for(i=0;i<n;i++)
    for(k=0;k<3;k++)
      velocidad[i][k]+=0.5*dt*fuerza[i][k];

  if(s_termalizacion) if(p%cada==0)
    termalizar(n, velocidad, temperatura, temperatura_inst);
}

if(sw[PRMS]) {
  if(!s_termalizacion) {
    if(sw[MODO]==MD) {
      if(sw[KP]) {
        energia_k_suma+=energia_k_inst;
        energia_k_prom=energia_k_suma/pev;
      }
      if(sw[VP]) {
        energia_p_suma+=energia_p_inst;
        energia_p_prom=energia_p_suma/pev;
      }
      if(sw[PP]) {
        w_suma+=w_inst;
        w_prom=w_suma/pev;
        presion_prom = densidad*(2*energia_k_prom+w_prom)/(3*n);
      }
      if(sw[TP]) {
        temperatura_prom = 2*energia_k_prom/(3*n);
      }
```

```
    }
    if(sw[MODO]==MC) {
      if(sw[VP]) {
        energia_p_suma+=energia_p_inst;
        energia_p_prom=energia_p_suma/pev;
      }
      if(sw[PP]) {
        presion_inst=calcular_presion(n,posicion,lado_caja,temperatura,
            densidad);
        presion_suma+=presion_inst;
        presion_prom=presion_suma/pev;
      }
    }

    //muestrear cada 10 pasos
    if(sw[G]&&p%10==0) {
      fdistradial(n,posicion,lado_caja,bins);
    }
  }
}

if(p==p_termalizacion) {
  s_termalizacion=0;

  if(sw[D]) {
    s_c_difusion=1;

    x=malloc(sizeof(vector)*(n+1));
    if(x==NULL)
      serror("No se ha podido alocar memoria para nuevas posiciones.");

    for(i=0;i<n;i++) {
      for(k=0;k<3;k++) x[i][k]=posicion[i][k];
    }
    difusion(n,x,it0,t0max,tmax,0);
  }
}

if(s_c_difusion&&(((int)pev)%cadad==0)) {
  difusion(n,x,it0,t0max,tmax,0);
}

// imprimir valores instantaneos y/o promedios seleccionados
if(cv>0) {
  printf("%5d %6.7g  ", p, t);
  param_orden = p_orden(n,posicion,lado_caja);
  printf("%e ",param_orden);
  if(sw[MODO]==MC) printf("%3.4g ", acep);
  for(j=0;j<(s_termalizacion?cvi:cv);j++) {
    printf("%e ", *pv[j]);
  }
  printf("\n");
}
```

```
    // para sacar el promedio del parametro de orden
    if(!s_termalizacion) {
      suma_s+=param_orden;
      veces_s++;
    }
  }

  if(sw[G]) {
    // indica finalizacion del muestreo
    fdistradial(n,NULL,densidad,0);
  }
  if(sw[D]) {
    difusion(n,NULL,it0,cadad,tmax,cadad*dt);
    free(x);
  }

  imprimir_posiciones(n,posicion);

  free(posicion);
  if(sw[MODO]==MD) {
    free(velocidad);
    free(fuerza);
  }
  free(pv);

  // imprimir al final: T, densidad, <T>, <P>, <S>
  printf("\n\n%e\t%e\t%e\t%e\n",temperatura,densidad,temperatura_prom,
      presion_prom,suma_s/veces_s);

  return 0;
}
```