Universidad Nacional Autónoma de México

Facultad de Ciencias

Separación quiral en un modelo de malla

T E S I S

que para obtener el título de

Físico

P R E S E N T A

Rafael Cruz Rodríguez

directora de tesis

Dra. Jacqueline Quintana Hinojosa

Ciudad Universitaria, Ciudad de México, 2019

Hoja de datos del jurado

1. Datos del alumno
Cruz
Rodríguez
Rafael
55 53 53 75 26
Universidad Nacional Autónoma de México
Facultad de Ciencias
Física
412003879

2. Datos de la asesora
Dra
Jacqueline
Quintana
Hinojosa

3. Datos del sinodal 1
Dr
David Philip
Sanders

4. Datos del sinodal 2
Dr
Ignacio Luis
Garzón
Sosa

5. Datos del sinodal 3
Dra
Roxana Mitzayé
del Castillo
Vázquez

6. Datos del sinodal 4
Dr
Francois Alain
Leyvraz
Waltz

7. Datos del trabajo escrito.
Separación quiral en un modelo de malla
(sin subtítulo)
139 p.
2019

Separación quiral en un modelo de malla

Rafael Cruz Rodríguez

22 de julio de 2019

A mi madre y a mi hermano.

Agradecimientos

Agradezco infinitamente a mi madre, Martha Rodríguez de los Santos, por hacerme quien soy y darme lo que tengo.

A mi hermano, Gustavo. Mi vida fue mejor desde que llegaste, pequeño gato.

Al resto de mi familia, por estar ahí cuando los he necesitado.

A mis amigas, mis amigos y a mi novia, por apoyarme siempre. A todos los quiero mucho.

Agradezco de manera muy especial a mi asesora, la Dra. Jacqueline Quintana Hinojosa, por todo el apoyo académico y no académico que me ha brindado durante estos más de dos años. Aprendí mucho bajo su tutela.

A mis sinodales, la Dra. Roxana del Castillo y los Dres. François Leyvraz, David Sanders e Ignacio Garzón por tomarse el trabajo de revisar y darme comentarios y correcciones de este trabajo.

A todos los profesores que me han tocado en la vida.

Al Instituto de Química de la UNAM por acogerme en sus instalaciones durante el desarrollo de esta tesis y por el apoyo otorgado mediante el Programa de Becas Internas 2018.

Al Consejo Nacional de Ciencia y Tecnología, por el apoyo otorgado a través del Proyecto de Investigación CONACYT 285502, con título "Efecto de la quiralidad, polaridad y anisotropía en el auto-ensamblaje molecular en dos dimensiones".

Finalmente estaré siempre agradecido con la Facultad de Ciencias y la Universidad Nacional Autónoma de México, por mi formación profesional y por ser mi segunda casa.

Índice de figuras

Índice de tablas

Índice general

Capítulo 1

Introducción

El propósito de este trabajo es investigar si es posible producir separación de fases enantioméricamente puras (separación quiral) en un sistema bidimensional cuyas interacciones son puramente energéticas, excluyendo los efectos de volumen excluido. Para esto se construyó un modelo sobresimplificado cuyas moléculas están situadas en una malla cuadrada. La quiralidad se expresa a través de 3 sitios atractivos situados en una forma molecular similar a la letra L. Véase la Fig. 1.1.

El antecedente de esto se encuentra en un estudio [16] en donde se consideraron solo las interacciones provenientes del volumen excluido, lo cual implica que el sistema esta fuera de red. Los estudios con este propósito realizados por el grupo de investigación han sido todos considerando solo el volumen excluido. Es decir, tomando en cuenta solo los efectos geométricos o entrópicos. En este sentido el considerar este tipo de interacciones (las de volumen excluido) hace más factible el proceso de separación quiral.

Debido a que el objetivo de este trabajo es analizar los efectos complementarios a los estudios anteriores, se usaron interacciones puramente energéticas. Cabe mencionar que esto puede implicar que el proceso de selección puede no tener tantos o efectos tan excluyentes y definitivos como el volumen excluido. Esto ocurre particularmente por haber utilizado el pozo cuadrado como sitios atractivos.

Para lograr el objetivo se aplicó el método de Metropolis Montecarlo en el ensamble canónico y gran canónico. Los movimientos de prueba consisten en cambiar la quiralidad de la partícula elegida. El parámetro de orden que permite cuantificar el fenómeno se define como el exceso de partículas de una especie homoquiral respecto de la otra, llamado **exceso enantiomérico**. En este trabajo también se trató de caracterizar la separación quiral observando el número y tamaño de cúmulos de una especie homoquiral, llamándola **microsegregación**.

A lo largo de este proyecto se han encontrado varios detalles del modelo y de su implementación para simulaciones numéricas que lo han vuelto un problema más complicado -e interesante- de lo que se pensó en un principio. Debido a lo cual este proyecto ha tomado mucho más tiempo del que se tenía pensado. Continuarlo lo extendería todavía más, por lo que se ha decidido presentar solamente los resultados parciales obtenidos hasta el momento. Aunque los resultados son parciales, dan información relevante sobre el comportamiento del modelo.

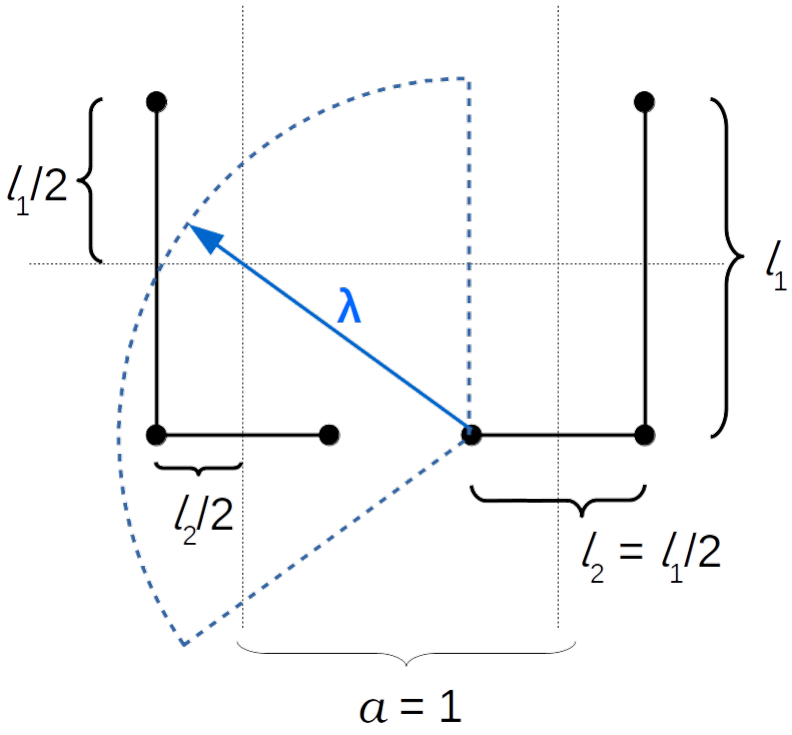


Figura 1.1: Partículas quirales en forma de L.

Capítulo 2

El modelo de Ising

En este capítulo se habla sobre el modelo de Ising, un modelo muy sencillo para estudiar materiales magnéticos y, específicamente, la transición ferromagnética–paramagnética. Se incluyen soluciones exactas y numéricas del modelo.

2.1. Introducción

Un área muy importante e interesante de estudio en física y materia condensada es el de materiales magnéticos. Se sabe que las propiedades magnéticas de un imán son originadas por los momentos magnéticos o por los espines de los átomos o moléculas que lo componen [1]. Los momentos interactúan con sus vecinos, por lo que al menos localmente la orientación de uno está influido por la orientación de los demás. La orientación también está influida por las fluctuaciones térmicas que presenta el material.

Al menos tres fenómenos magnéticos [2] son de interés en este trabajo: ferromagnetismo, antiferromagnetismo y paramagnetismo.

- El **ferromagnetismo** ocurre cuando los momentos magnéticos de los componentes de un material tienden a alinearse espontáneamente en la misma dirección y sentido. De esta manera, el material adquiere una magnetización neta distinta de cero.

- El **antiferromagnetismo** ocurre cuando los momentos tienden a alinearse en la misma dirección, pero en sentido opuesto al de sus vecinos. En este caso, la magnetización neta del material es muy cercano a cero.

- El **paramagnetismo** es un fenómeno que ocurre cuando los momentos magnéticos se alinean en presencia de un campo magnético externo. Los momentos tienden a alinearse paralelamente a dicho campo externo. Cuando desaparece el campo externo, los momentos recuperan el orden (macroscópico) que tenían.

En ausencia de alguno de estos fenómenos, los momentos magnéticos apuntan en direcciones arbitrarias, aunque influidos débilmente por las direcciones de sus vecinos. En algunos materiales aparecen parches de momentos con la misma orientación. Los parches son los que difieren entre sí

en orientación. Dado que los parches también son microscópicos, ambas situaciones causan que la magnetización neta del material sea prácticamente cero.

Las propiedades magnéticas de un material se pueden controlar mediante la temperatura y la acción de un campo magnético externo, entre otras maneras. Por una parte, existe una temperatura T_C (denominada temperatura de Curie) dependiente de cada material, por debajo de la cual el material adquiere orden microscópico y ocurre ferro o antiferromagnetismo; por encima de ella se pierde el orden de los momentos magnéticos. Por otra parte, un campo magnético puede inducir paramagnetismo, explicado anteriormente.

Para estudiar de manera teórica los materiales magnéticos es necesario modelar las interacciones entre momentos magnéticos y la influencia de la temperatura y el campo magnético externo sobre el material. También es necesario modelar la distribución espacial de los átomos; lo más común es estudiar materiales magnéticos sólidos, por lo que el material puede ser cristalino, cuasicristalino o amorfo. El modelo debe reproducir, al menos cualitativamente, los estados paramagnético a altas temperaturas, (anti)ferromagnético a bajas temperaturas y la transición continua pero rápida entre estados.

2.2. El modelo de Ising

El **modelo de Ising** [1] es un modelo muy sencillo de un material magnético. La versión más simple y estudiada consiste en una malla cuadrada (cúbica simple o su generalización a más dimensiones) de tamaño N sobre la cual hay espines s_i que "apuntan hacia arriba" (tiene espín $s_i = +1$) o "hacia abajo" (tiene espín $s_i = -1$). Los espines pueden interactuar con sus primeros vecinos de la malla. El hamiltoniano del sistema es

$$H = -J \sum_{\langle ij \rangle} s_i s_j - B \sum_i s_i,$$

donde J es una constante de interacción, $\sum_{<ij>}$ representa la suma sobre primeros vecinos $< ij >$ y B es un campo magnético externo. Una representación gráfica del modelo de Ising es la mostrada en la Fig. 2.1.

El signo de J induce ferromagnetismo o antiferromagnetismo. Cuando $J > 0$, los espines s_i y s_j deben tener el mismo signo para que se cumpla $-Js_is_j = -J < 0$, por lo que $J > 0$ favorece el comportamiento ferromagnético. Si $J < 0$, para que $-Js_is_j$ sea negativo entonces cada espín del par debe tener signo opuesto. De esta manera, el signo de un espín tiende a ser el opuesto del de sus vecinos, lo que en el sistema se refleja en la alternancia de signos entre espines. Esto es comportamiento antiferromagnético. En la Fig. 2.2 se esquematizan ambos comportamientos. Finalmente, en el caso $J = 0$ los espines no interactúan con sus vecinos.

De manera similar, el signo de B induce una u otra orientación en el sistema. Si $B > 0$, habrá una tendencia a que $s_i = 1$. Si $B < 0$, la tendencia será que $s_i = -1$. $B = 0$ indica ausencia de campo externo.

En este trabajo se muestra el comportamiento del modelo de Ising para el caso ferromagnético ($J > 0$) y en ausencia de campo magnético externo ($B = 0$). Con esto, el hamiltoniano se simplifica:

$$H = -J \sum_{<ij>} s_i s_j.$$

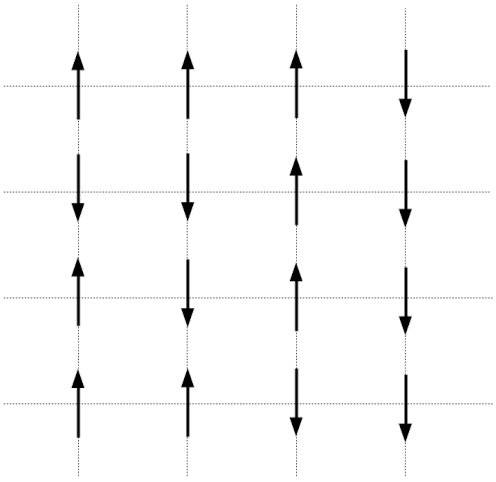
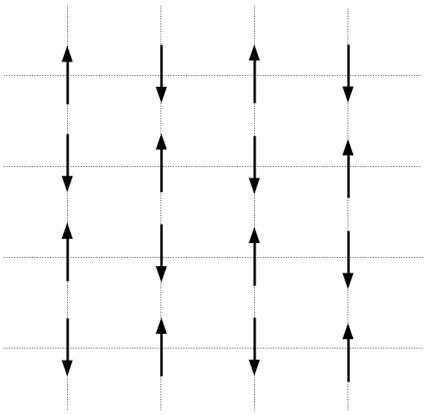
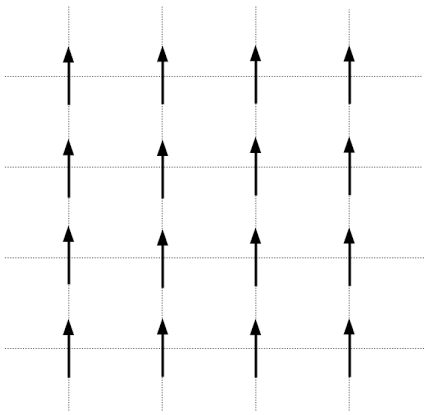


Figura 2.1: Representación gráfica del modelo de Ising en dos dimensiones. Las flechas representan los espines, mientras que las líneas punteadas representan la malla.



(a) Comportamiento antiferromagnético.



(b) Comportamiento ferromagnético.

Figura 2.2: Comportamiento ferromagnético y antiferromagnético en el modelo de Ising.

La función de partición del modelo en el ensamble canónico es

$$Z = \sum_{s_1=\pm 1} \sum_{s_1=\pm 2} \cdots \sum_{s_N=\pm 1} \exp\left(\beta J \sum_{<ij>} s_i s_j\right) = \sum_{\{s_i\}} \exp(-\beta H),$$

donde $\beta = 1/kT$ y $\{s_i\}$ denota el conjunto de los estados de cada espín s_i.

Ya que hay N espines y cada uno puede estar en 2 estados, el sistema tiene 2^N posibles configuraciones.

Dos de los parámetros más importantes a la hora de estudiar el modelo de Ising son la energía instantánea $E = H = -J\sum_{<ij>} s_i s_j$ y la magnetización instantánea $M = \sum_i s_i$. También es muy útil la magnetización instantánea por espín $m = M/N = \frac{1}{N}\sum_i s_i$. A partir de ellos se pueden calcular la energía promedio de los estados $\langle E\rangle = \langle H\rangle = \langle -J\sum_{<ij>} s_i s_j\rangle$, la magnetización promedio $\langle M\rangle = \langle \sum_i s_i\rangle$ y los promedios por espín $\langle e\rangle = \langle E\rangle/N$ y $\langle m\rangle = \langle M\rangle/N$.

Dado que, en el equilibrio, los valores de energía y magnetización que adquiere un sistema a lo largo de su evolución no son constantes, sino que fluctúan alrededor de un valor promedio, medir dichas fluctuaciones también puede ser relevante. En particular, las fluctuaciones de la energía se cuantifican con el **calor específico** por espín

$$c = \frac{k\beta^2}{N}(\langle E^2\rangle - \langle E\rangle^2)$$

y las de la magnetización con la **susceptibilidad magnética** por espín

$$\chi = \frac{\beta}{N}(\langle M^2\rangle - \langle M\rangle^2) = \beta N(\langle m^2\rangle - \langle m\rangle^2).$$

2.3. Soluciones exactas del modelo

La primera solución del modelo la dio precisamente Ising. Estudió la versión infinita del modelo en una dimensión, donde el modelo consiste en espines ubicados sobre una línea infinita. Encontró que existe magnetización espontánea únicamente para $T = 0$; para las demás temperaturas la magnetización neta es cero. Con esto supuso que el modelo no presentaba magnetización espontánea a temperatura distinta de cero para cualquier dimensión mayor a 1.

Sin embargo, años después Onsager [3] demostró que el modelo en una malla bidimensional infinita sí presenta magnetización espontánea para $T \neq 0$. Específicamente encontró que el comportamiento de la energía en todo el rango de temperaturas es

$$\epsilon(T) = -2J\tanh(2\beta J) + \frac{K}{2\pi}\frac{\mathrm{d}K}{\mathrm{d}\beta}\int_0^\pi \mathrm{d}\phi \frac{\mathrm{sen}^2(\phi)}{\Delta(1+\Delta)},$$

donde

$$K(\beta, J) = \frac{2}{\cosh(2\beta J)\coth(2\beta J)}$$

$$\Delta = \sqrt{1 - K^2\mathrm{sen}(\phi)}$$

El comportamiento de la magnetización revela más información sobre el modelo, que se muestra gráficamente en la Fig. 2.3. En el rango $[0, T_C]$ es

$$\langle m\rangle = [1 - \sinh(2/T)^{-4}]^{1/8} \tag{2.1}$$

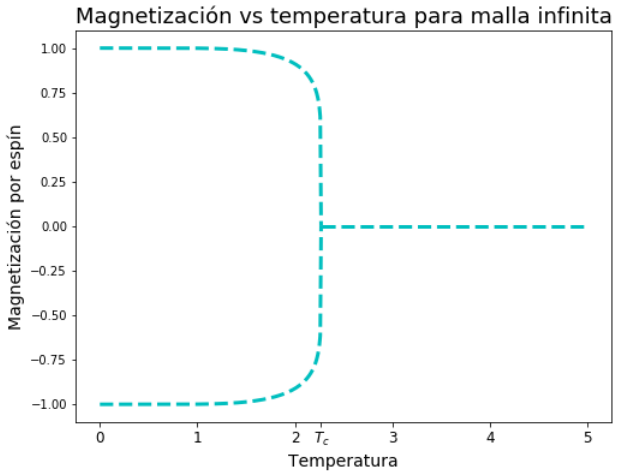


Figura 2.3: Magnetización como función de la temperatura para una malla bidimensional infinita. Resultado obtenido por Onsager.

y arriba de T_C se cumple que $m = 0$. La temperatura a la que ocurre la transición de paramagneto a ferromagneto es $T_C = \dfrac{2J/k}{\log(1+\sqrt{2})} \approx 2.3J/k$. Al bajar la temperatura y llegar a $T = T_C$, $\langle m \rangle$ la magnetización cambia de 0 a ± 1 en un pequeño intervalo de valores de manera continua.

Los dos estados base del modelo de Ising a $T = 0$ consisten en el sistema con todos los espines apuntando en el mismo sentido. Como hay dos posibles orientaciones, hay dos estados base. Ambos estados son simétricos (basta con cambiar la orientación de cada espín para pasar de un estado a otro) por lo que sus energías son iguales. Para $T \gg T_C$, los estados más comunes son en los que los espines apuntan aleatoriamente, de tal manera que la magnetización neta es nula.

En tres y más dimensiones, con mallas infinitas, no se ha hallado solución exacta al modelo. Sin embargo, hay varias soluciones aproximadas obtenidas gracias tanto a métodos teóricos de aproximación como a métodos numéricos. Básicamente el único método numérico utilizado es el de Montecarlo.

2.4. Soluciones numéricas con el método de Montecarlo

Debido a que el modelo de Ising no incluye ecuaciones de Newton, no es posible simular el sistema de manera determinista. La única manera de simularlo es utilizando algún método probabilístico, como el método de Montecarlo (del que se habla en el Capítulo 4). Además, a diferencia de los métodos teóricos utilizados por Ising y Onsager, en una simulación no se pueden estudiar sistemas infinitos. Una solución muy común es estudiar sistemas con condiciones periódicas, donde los espines situados en un extremo de la malla (por ejemplo, el extremo izquierdo) interactúan con los del otro extremo (el derecho). Afortunadamente y a pesar de las diferencias entre teoría y simulaciones, los resultados de las simulaciones sí pueden proveer de mucha información importante acerca del modelo de Ising.

El método de Montecarlo aplicado al modelo de Ising es el siguiente:

1. Inicializar el sistema con la orientación de los espines distribuida aleatoriamente. También es posible fijar un estado inicial con todos los espines con la misma orientación para temperaturas bajas, ya que se sabe que los estados base a esas temperaturas tienen esa configuración o una muy similar.

2. Elegir un espín aleatoriamente e invertir su orientación.

3. Calcular la diferencia de energías entre los estados inicial y de prueba. Este paso se puede simplificar en el modelo de Ising: si la energía de interacción del espín con sus vecinas era U_o, su nueva energía es $U_t = -U_o$, por lo que la diferencia es $\Delta U = -2U_o$.

4. Calcular la probabilidad de aceptar la modificación del sistema con

$$p = \text{mín}\left\{1, \exp(-\Delta U/k_b T)\right\}.$$

5. Una vez aceptada o rechazada la modificación, se calcula la nueva energía del sistema y se repite el proceso desde el paso 2.

Como se menciona más adelante en el Apéndice de Equibración (Apéndice A) , primero se debe dejar correr la simulación un cierto número de pasos antes de comenzar a medir las propiedades de interés.

Para este trabajo se realizaron simulaciones del modelo de Ising en una malla de 50×50 con condiciones periódicas y se midieron la energía y la magnetización. En el caso de la magnetización, se compara con el comportamiento teórico de la malla infinita. Se fijaron $J = 1$ y $k = 1$ para simplificar la simulación.

En las Figuras 2.4 y 2.5 se muestran los resultados obtenidos en las simulaciones. En ambos casos se puede observar que en un intervalo alrededor de T_C cambian notoriamente los valores de los parámetros estudiados. En el caso de la energía, crece de $E = 2$ para temperaturas cercanas a cero a $T = 0$ para temperaturas altas (aunque en el rango mostrado solo llegue a alrededor de $E = 0.4$). El cambio es más rápido alrededor de $T = T_C$. La magnetización tiene un comportamiento más interesante. En concordancia con la teoría, para $T < T_C$ algunas simulaciones terminaron con magnetización neta positiva y otras con negativa. Cerca de T_C, la magnetización sufre un decrecimiento en su valor absoluto hasta que se vuelve casi cero en $T \approx 2.5$ y así se mantiene para cualquier temperatura más alta. El comportamiento de la magnetización de la malla finita alrededor de T_C no es exactamente igual al de la malla infinita, pero sí tiene un gran parecido.

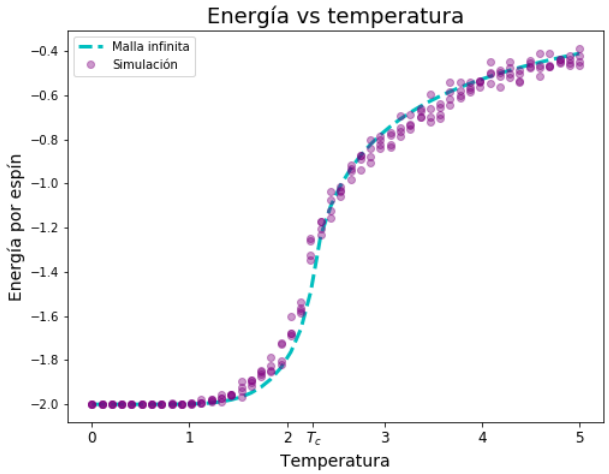


Figura 2.4: Energía contra temperatura obtenida de una simulación de una malla de 50×50. Pasa de $E = -2$ cuando $T = 0$ a $E = 0$ cuando $T \to \infty$. La línea azul claro muestra el resultado para la malla infinita, los puntos púrpuras los datos de la simulación.

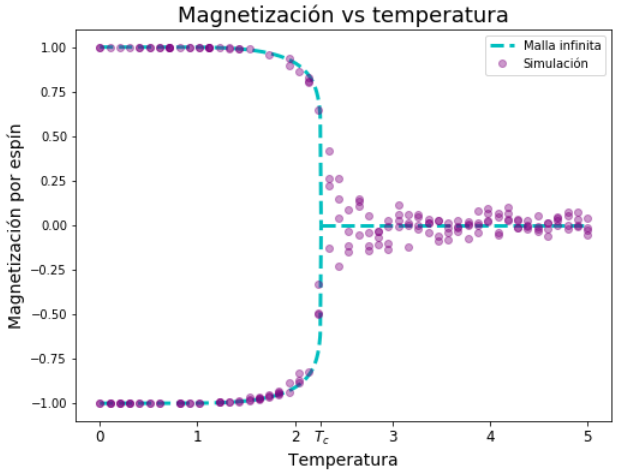


Figura 2.5: Magnetización contra temperatura obtenida de una simulación de una malla de 50×50. Pasa de $m = \pm 1$ cuando $T = 0$ a $m = 0$ cuando $T \to \infty$. La línea azul claro muestra el resultado para la malla infinita, los puntos púrpuras los datos de la simulación.

Capítulo 3

Quiralidad y separación quiral en química

En este capítulo se habla sobre la propiedad geométrica de la quiralidad, su importancia para los seres vivos y la química orgánica, la obtención de moléculas quirales y algunos estudios que se han hecho para entender la segregación espontánea de mezclas de moléculas quirales.

3.1. Quiralidad

La **quiralidad** [4] es la propiedad que posee algún objeto cuya imagen especular (la obtenida con un espejo) no es superponible a la original y viceversa, ya que la imagen espejo de la primera imagen es el objeto original. Por ejemplo, la imagen especular de la mano derecha es la mano izquierda. Sin embargo, al intentar superponer la izquierda sobre la derecha, se observa que no encajan perfectamente.

Como la imagen especular de la imagen original vuelve a ser la original, solo puede haber dos objetos relacionados por reflejo. A ambos objetos se les denomina **enantiomorfos**. La quiralidad es una propiedad estudiada en geometría, física, química y biología.

Un ejemplo de molécula quiral es el bromocloroyodometano (Fig. 3.1), que tiene estructura tetraédrica. La línea punteada es un enlace que apunta hacia "adentro"; la línea gruesa apunta hacia "afuera".

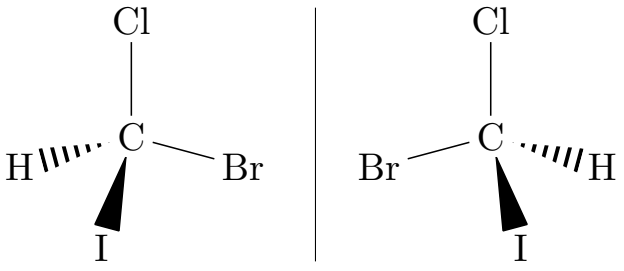


Figura 3.1: Moléculas quirales de bromocloroyodometano.

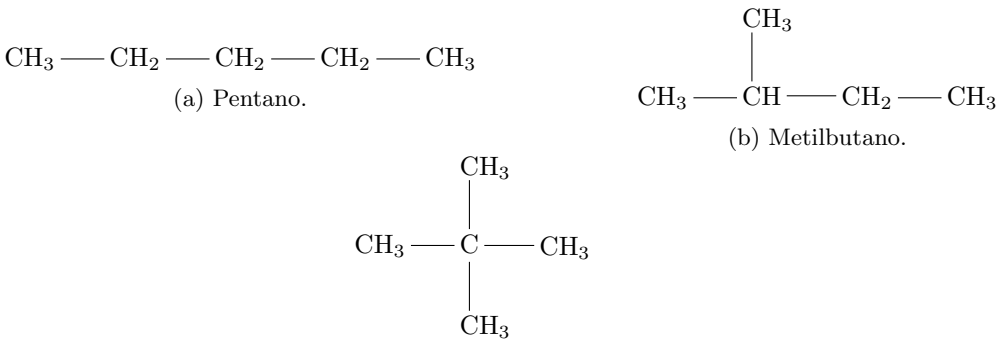
3.2. Isómeros, estereoisómeros y enantiómeros

Para entender el papel de la quiralidad en la química y la biología, primero se debe hablar sobre algunos tipos de moléculas. Los **isómeros** son sustancias que poseen la misma composición y peso molecular pero difieren en sus propiedades. A nivel molecular, dichas sustancias poseen el mismo número y tipo de átomos pero difieren en estructura [4]. Los isómeros se pueden clasificar en dos tipos. Los **isómeros estructurales** difieren entre sí por su **conectividad**, lo que quiere decir que para pasar de un isómero estructural a otro es necesario "desconectar" algunos átomos de su lugar y "conectarlos" en otro lugar. Los isómeros que además poseen la misma conectividad pero difieren en su arreglo tridimensional se denominan **estereoisómeros**.

Un ejemplo de isomería es la existente entre el pentano, el metilbutano y el dimetilpropano. Los tres tienen la fórmula química C_5H_{12} pero las estructuras de las tres moléculas son distintas (véase la Fig. 3.2). Además se trata de isomería estructural, ya que para llegar de uno a otro hay que "desconectar" algunos carbonos e hidrógenos y "conectarlos" en otros átomos. Por ejemplo, para pasar del pentano al metilbutano se puede desconectar el grupo metilo (el CH_3) de extrema derecha y un hidrógeno del segundo carbono, y conectar el hidrógeno en el tercer carbono (para que se vuelva CH_3) y conectar el metilo en el segundo carbono.

A su vez, los estereoisómeros se dividen en enantiómeros y diastereoisómeros. Se dice que dos estereoisómeros son **enantiómeros** si una de ellas es la imagen especular no superponible de la otra, lo que significa que estas moléculas son quirales. No es sorprendente que los enantiómeros posean propiedades físicas y químicas (escalares) idénticas, ya sea punto de fusión y ebullición, solubilidad, densidad, etc., o a lo más difieran en un signo, como en la actividad óptica. Un ejemplo es el ya mostrado en la Fig. 3.1. Los **diastereoisómeros** o **diastereómeros**, por otra parte, son estereoisómeros no quirales. Por ejemplo, el ácido (Z)-3-amino-2-butenoico y el ácido (E)-3-amino-2-butenoico (véase la Fig. 3.3) tienen exactamente los mismos átomos y para pasar de una molécula a otra no es necesario desconectarla, solo basta "torcerla" (en un sentido abstracto, porque químicamente sí es necesario romper el doble enlace). Por lo tanto, son estereoisómeros. Sin embargo, no son la imagen espejo de la otra, por lo que se trata de diastereoisómeros. A diferencia de los enantiómeros, pueden haber varios diastereoisómeros relacionados.

La quiralidad (o la ausencia de ésta) es una propiedad global de una molécula, pero normalmente es posible encontrar algunos átomos (en particular, de carbono) con los cuales, a partir de la



(a) Pentano.

(b) Metilbutano.

(c) Dimetilpropano.

Figura 3.2: Ejemplos de hidrocarburos que son isómeros estructurales.

(a) Ácido (Z)-3-amino-2-butenoico. (b) Ácido (E)-3-amino-2-butenoico.

Figura 3.3: Ejemplos de diastereoisómeros.

configuración espacial de sus vecinos, es posible determinar si una molécula es quiral o no. A estos átomos (específicamente a los sitios que ocupan en la molécula) se les conoce como **centros quirales** o **centros de quiralidad**. Los enantiómeros pueden poseer uno o más centros de quiralidad, mientras que los diastereoisómeros, cuando tienen centros de quiralidad, tienen más de uno. Es importante resaltar que no siempre existen centros de quiralidad, pero es lo más común.

3.3. Clasificación de los enantiómeros

Ya que los enantiómeros poseen exactamente la misma fórmula química, es necesaria alguna notación para referirnos a cada uno de ellos. La regla de quiralidad CIP [4, 5] sirve para calcular la configuración absoluta de los centros quirales en una molécula. La regla clasifica los centros quirales en R (del latín *rectus*, derecha) y S (del latín *sinister*, izquierda), asignando prioridades a los elementos ligados al centro quiral. Si los elementos se denominan A, B, D y E y suponiendo que la secuencia de prioridad es A > B > D > E, el modelo molecular (tetraédrico) se observa desde el lado opuesto al elemento de menor prioridad, el E. Hecho esto, los elementos A, B y D se ven de una manera tripodal. Si estos elementos están ordenados en sentido horario (A $\rightarrow$ B $\rightarrow$ D), al centro quiral se le asigna la letra R. Si van en sentido antihorario (D $\rightarrow$ B $\rightarrow$ A), se le asigna la letra S. En el caso de que haya más centros quirales, se escribe la posición del centro quiral junto a su quiralidad.

En general, la prioridad de cada átomo está dado por su número atómico. A mayor número atómico, mayor prioridad se le asigna al átomo. Si solamente hay tres átomos ligados, se asume que hay un átomo de número atómico cero en la posición donde debería ir. En el caso de que haya dos átomos iguales unidos al centro quiral, se comienza a explorar los átomos siguientes, asignando prioridades a cada rama hasta determinar de manera unívoca la de mayor prioridad. Hay más reglas en caso de que haya múltiples enlaces, existan anillos y algunos casos más.

Por ejemplo, para clasificar una molécula de bromocloroyodometano en R o S se procede como sigue:

- Se ordenan los elementos de la molécula por número atómico. Los números atómicos son: H: 1; Cl: 17; Br: 35; I: 53. Entonces el orden A > B > D > E es I > Br > Cl > H.
- Se observa el modelo molecular desde el lado opuesto al elemento de menor prioridad, que en este caso es el hidrógeno. En la molécula del lado izquierdo de la Fig. 3.4, el orden atómico I $\rightarrow$ Br $\rightarrow$ Cl va en sentido horario, por lo que se trata del enantiómero R. En la molécula izquierda, el orden atómico va en sentido antihorario, así que se trata del enantiómero S.

Otra clasificación muy usada es la (+)-(-). Esta clasificación se basa en el hecho de que las moléculas quirales rotan el plano de polarización de la luz. El símbolo (+) denota que las moléculas

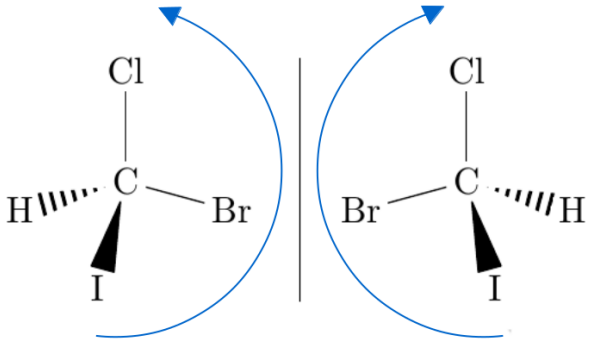


Figura 3.4: Clasificación R-S del bromocloroyodometano. La molécula del lado izquierdo es el enantiómero R, la del lado derecho es el S.

rotan el plano hacia la derecha (dextrorotatorias); el símbolo (-) indica que lo hacen hacia la izquierda (levorotatorias). También se suelen usar los símbolos (d)-(l), pero se desaconseja su uso por su parecido con la clasificación D-L, explicada después. El uso de la clasificación (+)-(-) presenta el problema de que la dirección en la que la sustancia rota el plano de polarización de la luz no es única. Depende de la longitud de onda, de la temperatura, del solvente y de la concentración. Además, impurezas con actividad óptica mayor a la de la sustancia a estudiar también pueden alterar de manera significativa los resultados. Por último, las otras dos clasificaciones se refieren a la configuración absoluta de las moléculas, mientras que la actividad óptica no tiene relación alguna con la configuración de la molécula excepto que sea quiral.

Para los carbohidratos y aminoácidos existe la clasificación D-L [6], con las letras en versalitas, basada en la proyección planar (de Fischer) de las moléculas. Para estas moléculas esta clasificación es muy conveniente, ya que en vez de nombrar todos los centros quirales más el nombre completo, simplemente se usa una letra y su nombre común. Por ejemplo, D-glucosa en vez de (2R,3S,4R,5R)2,3,4,5,6-pentahidroxihexanal.

3.4. Homoquiralidad y efectos en los seres vivos

Una gran cantidad de moléculas orgánicas son enantioméricas. Más importante aún es el hecho de que, por una parte, en laboratorio suelen sintetizarse ambos enantiómeros en igual proporción y mezclados (a las mezclas de enantiómeros con la misma proporción de ambos se les denomina **mezclas racémicas**). Por otra parte, los seres vivos suelen producir o asimilar adecuadamente uno solo de los enantiómeros: es el fenómeno de la **homoquiralidad**. Se sabe que casi todos los aminoácidos quirales son L-enantiómeros, mientras que los azúcares son D-enantiómeros [6]. Aún no existe un consenso sobre cómo surgió dicho fenómeno, aunque existen diversas teorías [4, 7], y si podría tener alguna ventaja con respecto a utilizar ambos enantiómeros (heteroquiralidad). Lo que es bien conocido es que, en un ser vivo, reemplazar unos enantiómeros por otros suele generar reacciones muy distintas que pueden ser incluso fatales. Las moléculas enantioméricas solo poseen efectos distintos cuando se encuentran en medios quirales. Como los seres vivos estamos compuestos de muchos enantiómeros de un solo tipo, somos medios quirales.

Saber si una molécula es quiral resulta muy importante a la hora de estudiar sus efectos en los seres vivos ya que, como se mencionó, es común que las reacciones sean muy distintas. Esto es especialmente necesario a la hora de desarrollar nuevos fármacos, como el desafortunado caso de la talidomida [8] lo demostró. La compañía alemana Chemie-Grünenthal descubrió que la talidomida, no siendo un barbitúrico (los barbitúricos son una familia de sedantes con alto potencial de adicción, derivados del ácido barbitúrico), se podía usar como un sedante no adictivo. Después se descubrió que también servía com antiemético (impide o alivia el vómito y las náuseas). A finales de la década de 1950, la talidomida se comenzó a recetar para mitigar las náuseas de mujeres embarazadas. Poco después aumentaron los casos de neuropatía periférica (daño a los nervios periféricos, los que comunican el cerebro y la médula espinal con el resto del cuerpo). En 1961, dos estudios independientes confirmaron que el consumo de talidomida durante el embarazo causa malformaciones en los recién nacidos. Después de distribuirse en 46 países y causar malformaciones en más de 10,000 recién nacidos antes de su retiro del mercado entre 1961 y 1962, el uso de talidomida se clasificó como el mayor desastre médico causado por el ser humano.

Las investigaciones encontraron que la talidomida es en realidad una molécula quiral [8, 9]. El *R*-enantiómero produce los efectos sedantes y antieméticos. El *S*-enantiómero, por otra parte, es el que tiene efectos teratogénicos (causa malformaciones en fetos). Además, un enantiómero de la talidomida se puede convertir en el otro bajo ciertas condiciones. Después del desastre, se impusieron pruebas mucho más estrictas a la industria farmacéutica para poder comercializar nuevos fármacos.

3.5. Obtención de sustancias enantioméricamente puras en laboratorio

Por todo lo mencionado, resulta de gran relevancia para la ciencia y para diversas industrias la obtención de compuestos enantioméricamente puros, además de la extracción de éstos de seres vivos. En laboratorio existen dos maneras de lograrlo [4]:

- Resolución de mezclas racémicas. Se busca separar los enantiómeros que conforman una mezcla racémica. Principalmente se logra utilizando métodos físicos (básicamente, cristalización de los enantiómeros) y químicos (la mayoría consiste en convertir la mezcla racémica en una mezcla de diastereoisómeros y aprovechar sus distintas propiedades físicas). De todos los compuestos enantioméricos conocidos, únicamente alrededor del 10 % se han podido sintetizar mediante métodos físicos [4].

- Síntesis enantioselectiva o asimétrica. Consiste en obtener un solo enantiómero (o una proporción mucho más alta de éste) a partir de un compuesto aquiral. Esto se logra utilizando reactivos, superficies, cristales o cristales líquidos quirales.

A la separación física o cristalización de mezclas racémicas en sustancias de un solo enantiómero también se le conoce como **separación quiral** o **segregación quiral** (espontánea). Se considera separación *espontánea* porque no se interactúa con los enantiómeros a separar a través de otras moléculas, si no a través de la temperatura, la presión, etcétera. Son los mismos enantiómeros los que se organizan para separarse.

3.6. Investigaciones teóricas

También hay investigaciones teóricas para descubrir con qué tipo de potenciales intermoleculares y bajo qué condiciones se estimula la resolución espontánea de mezclas racémicas.

Existen moléculas que en tres dimensiones no son quirales, pero en dos dimensiones sí lo son. A esta propiedad se le llama **proquiralidad**. Hay experimentos que muestran que al aplicar alguna restricción al sistema, como una reducción en la dimensionalidad, puede favorecer la separación quiral. Otro factor que parece favorecer la separación quiral es el aumento de la densidad de un sistema [15].

Huckaby et al. han estudiado diversos modelos en malla en dos y tres dimensiones, como el modelo de Andelman – de Gennes [13] y un modelo de moléculas con estructura $C(AB)_2$ [14].

Quintana et al. han estudiado un modelo de partículas en forma de zigzag [15] (tres segmentos de recta), en el cual las moléculas interactúan mediante potenciales infinitamente repulsivos y el sistema está limitado en dos dimensiones. Otro modelo estudiado [16] consiste en partículas en forma de "palo de hockey"(dos segmentos), del que se habla más adelante. Ambos modelos presentan separación quiral controlada principalmente por la densidad (y no por la temperatura, ya que son modelos atermales).

Capítulo 4

Métodos de Montecarlo

Una manera de estudiar un sistema macroscópico es realizando simulaciones numéricas de sus componentes microscópicos: átomos, iones, moléculas, etc., con mayor o menor detalle de los potenciales de interacción entre los distintos componentes. El número de partículas N puede ir desde las decenas hasta los cientos de millones, por lo que hay muchos grados de libertad. Como las propiedades de los sistemas macroscópicos resultan ser promedios de sus propiedades microscópicas, las simulaciones con muchas partículas se realizan para obtener dichos promedios: si H es el hamiltoniano clásico del sistema y A es una **observable** (una función de las variables microscópicas del sistema), el **promedio de** A en el ensamble canónico (a temperatura T, volumen V y número de partículas N fijos) es [11]

$$\langle A\rangle = \frac{\int_V A(\mathbf{p}^N,\mathbf{r}^N)\exp\big(-H(\mathbf{p}^N,\mathbf{r}^N)/k_BT\big)\mathrm{d}\mathbf{p}^N\mathrm{d}\mathbf{r}^N}{Z(T,V,N)},$$

donde $Z(T,V,N)=\int_V \exp\big(-H(\mathbf{p}^N,\mathbf{r}^N)/k_BT\big)\mathrm{d}\mathbf{p}^N\mathrm{d}\mathbf{r}^N$ es la **función de partición** del sistema.

Los conocidos métodos deterministas de integración numérica (como las fórmulas de Newton-Cotes o las cuadraturas de Gauss) son útiles cuando hay pocas variables, pero se vuelven muy ineficientes para más dimensiones. En esos casos, el método de Montecarlo* [11, 12, 10] es mucho más útil. Dicho método consiste en aproximar una integral de múltiples variables a través de puntos aleatorios del espacio multidimensional de integración.

4.1. Método de Montecarlo en el ensamble canónico

En principio, para realizar el cálculo de $\langle A\rangle$, el espacio donde se integra es el espacio fase del sistema físico a simular. Sin embargo, en la práctica el método de Montecarlo integra sobre el espacio configuracional.

Si se considera un hamiltoniano separable del tipo $H=K(\mathbf{p}^N)+U(\mathbf{r}^N)$, donde $K=\sum_i^{3N}\frac{p_i^2}{2m_i}$ es la parte cinética y U la parte potencial, la parte cinética de $\langle A\rangle$ se puede resolver analíticamente. De esta manera solo hace falta resolver la parte configuracional del promedio.

*En español lo correcto es escribir *Montecarlo*, a diferencia del inglés, donde se escribe *Monte Carlo*.

Si el sistema consta de N partículas, cada una con f grados de libertad posicionales (y orientacionales, si aplica), el espacio configuracional del sistema tiene Nf dimensiones. Como la parte cinética ya está resuelta, el estado completo del sistema (en una simulación de Montecarlo) se traduce en un punto del espacio configuracional, por lo que una simulación se representa como una serie de puntos en dicho espacio. Un **ensamble** de partículas (el conjunto de microestados accesibles al sistema, dadas ciertas restricciones) se representa como un subconjunto en el espacio configuracional.

El método de Montecarlo fue pensado originalmente para el ensamble canónico, cuyas variables naturales son el número de partículas N, el volumen V y la temperatura T. Ya que normalmente lo que interesa es comparar sistemas con N y V fijas (donde V puede ser infinito), la variable de mayor interés es T.

Si el sistema está descrito por el hamiltoniano H, la probabilidad de que el sistema se encuentre en un estado σ con temperatura T (o de manera equivalente, la probabilidad de que a T fija el estado del sistema sea tal que su hamiltoniano valga $H = E$) es proporcional a la **distribución de Boltzmann**:

$$P(\sigma, T) = \exp(-E(\sigma)/k_B T)/Z(T)$$

donde k_B es la constante de Boltzmann. La distribución de Boltzmann indica que, a T fija, los estados que ocurren con mayor frecuencia son los de menor energía.

El algoritmo básico del método de Montecarlo, llamado **algoritmo de Metropolis** [11, 12] es el siguiente:

Algoritmo de Metropolis

1. Inicializar el sistema, usualmente con una configuración espacial, orientacional, etc., al azar y calcular su energía U_{sist}.

2. Escoger una partícula o molécula de manera uniformemente aleatoria.

3. Intentar una modificación en el sistema: en el modelo de Ising (del cual se habla en el Capítulo 2) consiste en cambiar de signo un espín y en el modelo de partículas L, cambiar la quiralidad de una partícula. En otros casos consiste en un pequeño desplazamiento, rotación o conformación de la partícula o molécula elegida.

4. Calcular la diferencia de energías potenciales entre la configuración actual y la de prueba: $\Delta U = U_{\text{prueba}} - U_{\text{actual}}$. Normalmente solo hace falta calcular ΔU como el cambio en la energía de la partícula o molécula modificada.

5. Si $\Delta U \leq 0$, la modificación es aceptada. Si no, se escoge un número $\xi \in [0, 1]$ al azar y se compara con $\exp(-\Delta U/k_B T)$. En caso de que $\xi < \exp(-\Delta U/k_B T)$, la modificación es aceptada. En caso contrario es rechazada.

6. Una vez aceptada o rechazada la modificación, se calcula la nueva energía del sistema. Si el movimiento fue aceptado, la nueva energía es $U_{\text{sist}} = U_{\text{sist}} + \Delta U$. Si fue rechazado, la energía no cambia.

7. Se repite el proceso desde el paso 2.

Cada cierto número de repeticiones se calculan las propiedades de interés del sistema, para calcular al final los promedios del ensamble.

Es importante remarcar que, debido a que los cambios realizados al sistema a simular son estocásticos en vez de deterministas, el tiempo físico no es un parámetro que necesariamente se pueda introducir u obtener de la evolución de la simulación. Como consecuencia, la trayectoria que sigue la simulación en el espacio fase no necesariamente corresponde con la que seguiría el sistema físico o una simulación determinista. En particular, con el método de Montecarlo es posible permitir movimientos físicamente imposibles (como hacer que una partícula se mueva a un hueco sin que esté muy cerca de él) pero que generan estados representativos del sistema de manera más rápida.

El algoritmo permite que existan fluctuaciones en la energía. Sin embargo, entre más grande sea la fluctuación, es más probable que sea suprimida (porque $\exp(-\Delta U/k_BT) \to 0$ cuando $\Delta U \to \infty$). Así es como el algoritmo puede realizar un muestreo (usualmente) eficiente del espacio configuracional del sistema.

Normalmente el algoritmo se repite un determinado número de veces antes de medir los parámetros deseados, para permitir que el sistema "se termalice" o se equilibre. En el Apéndice A se habla más a fondo sobre este asunto, pero lo más importante es lo siguiente. Cuando se estudia un nuevo sistema y se posee poca información sobre su estructura, lo más común es inicializar la simulación en una configuración arbitraria, que no necesariamente es una configuración de equilibrio. Conforme avance la simulación, las configuraciones serán más representativas del sistema en equilibrio.

4.2. Método de Montecarlo en el ensamble gran canónico

A pesar de que el método de Montecarlo fue ideado en el ensamble canónico, se puede adaptar a más ensambles. Un ejemplo es la adaptación al ensamble gran canónico. En este caso, las variables del sistema son la energía U, la presión P y el número de moléculas N. Ahora se tienen fijas la temperatura T, el volumen V y el potencial químico μ.

En el ensamble gran canónico, el promedio de una función A dependiente de las variables microscópicas del sistema es [11]

$$\langle A\rangle = \frac{\sum_{N=0}^{\infty}(N!)^{-1}V^N z^N \int_V A(\mathbf{p}^N,\mathbf{r}^N)\exp\big[-H(\mathbf{p}^N,\mathbf{r}^N)/k_BT\big]\mathrm{d}\mathbf{p}^N\mathrm{d}\mathbf{r}^N}{Q(T,V,\mu)},$$

donde

$$Q(T,V,\mu) = \sum_{N=0}^{\infty}(N!)^{-1}V^N z^N \int_V \exp\big[-H(\mathbf{p}^N,\mathbf{r}^N)/k_BT\big]\mathrm{d}\mathbf{p}^N\mathrm{d}\mathbf{r}^N$$

es la función de partición del sistema en el ensamble gran canónico. Además se definen las variables $\Lambda = \left(\dfrac{h^2}{2\pi m k_B T}\right)^{1/2}$ y $z = \dfrac{\exp(\mu/k_BT)}{\Lambda^3}$ como la longitud de onda de de Broglie y la actividad, respectivamente. En estas simulaciones es necesario calcular N con cada paso de Montecarlo.

Uno de los algoritmos para las simulaciones de Montecarlo en el gran canónico consiste en inicializar aleatoriamente el sistema, calcular su energía inicial U_{sist} y repetir el ciclo de elegir aleatoriamente uno de los siguientes tres movimientos:

1. Modificar el estado de una partícula. Es equivalente al paso 3 del algoritmo de Montecarlo en el ensamble canónico. Obtener ΔU y calcular la probabilidad de aceptar la modificación con la misma probabilidad que en el caso del ensamble canónico,

$$p = \text{mín}\big\{1, \exp(-\Delta U/k_bT)\big\}.$$

2. Destruir una partícula y obtener ΔU. En caso de intentar destruir una partícula, la probabilidad de que se acepte el movimiento por el método ya mencionado está dado por

$$p = \text{mín}\left\{1, \exp\left(-\Delta U/k_B T + \ln\frac{N}{zV}\right)\right\}.$$

3. Crear una partícula (preferentemente en un espacio con pocas o ninguna partícula) y obtener ΔU. La probabilidad que utiliza el algoritmo de Metropolis para crear una nueva partícula en el sistema es

$$p = \text{mín}\left\{1, \exp\left(-\Delta U/k_B T + \ln\frac{zV}{N+1}\right)\right\}.$$

Una vez aceptado o rechazado cada modificación, se actualiza U_{sist} y se continúa con el ciclo.

Al igual que en el caso del método en el ensamble canónico, cada cierto número de repeticiones se calculan las propiedades de interés del sistema.

Para asegurarse de que la simulación se realiza correctamente, se impone que la probabilidad de eliminar una partícula sea la misma que la probabilidad de insertarla.

4.3. Objetivo

En este trabajo se presenta un modelo de moléculas quirales en forma de L situadas en una malla bidimensional para averiguar bajo qué condiciones produce separación quiral. Para ello se implementa el modelo de estudio en un programa computacional y se realizan simulaciones de Montecarlo en los ensambles canónico y gran canónico. Los detalles del modelo y de la implementación de Montecarlo se encuentran en el Capítulo 5.

Capítulo 5

Modelo de partículas en forma de L e implementación

5.1. Sistema de partículas en forma de L

El modelo de estudio consiste en N partículas quirales bidimensionales ubicadas en una red cuadrada con condiciones de frontera periódicas. La longitud reducida de las aristas de la red se fijó en $a = 1$. (Nota: En el Apéndice B se habla sobre unidades reducidas. A menos que se mencione lo contrario, todas las unidades mencionadas a lo largo del trabajo son unidades reducidas.) Cada partícula ocupa un sitio de la red. Las partículas tienen la forma de la letra L: dos segmentos de recta unidos con un ángulo recto. Por simplicidad se fija la longitud del segmento largo l_1 como el doble de la longitud del segmento corto l_2: $l_1 = 2l_2$. En la simulaciones se fija $l_1 = 0.9$, por lo que $l_2 = 0.45$. Todas las partículas están alineadas como lo muestra la Fig. 5.1. Las aristas de la red se muestran como líneas punteadas (negras) que bisectan cada segmento de las partículas. Los movimientos posibles de cada partícula son cambiar su quiralidad y desplazarse por los sitios de la malla (solo si hay sitios vacíos en la malla). Las partículas no pueden rotar en sus posiciones. En la siguiente sección se especifica cómo ocurren estos movimientos.

Cada partícula posee tres sitios de interacción atractivos localizados en sus vértices. El potencial que actúa entre los sitios de interacción de diferentes partículas es el **pozo cuadrado** U_{PC}

$$U_{\mathrm{PC}}(r) = \begin{cases} \infty, & r = 0 \\ -\epsilon, & 0 < r < \lambda \\ 0, & \lambda < r \end{cases}$$

donde ϵ es la profundidad del pozo y λ es el **rango de interacción** del potencial. Para el estudio de este modelo se fijó $\epsilon = 1$ y λ varía para cada sistema.

A partir de la descripción del modelo se puede escribir su hamiltoniano

$$H(\sigma) = \sum_{i=1}^{N} \sum_{\substack{j=1 \\ j \neq i}}^{N} \sum_{p_i=1}^{3} \sum_{q_j=1}^{3} U(r_{p_i,q_j}),$$

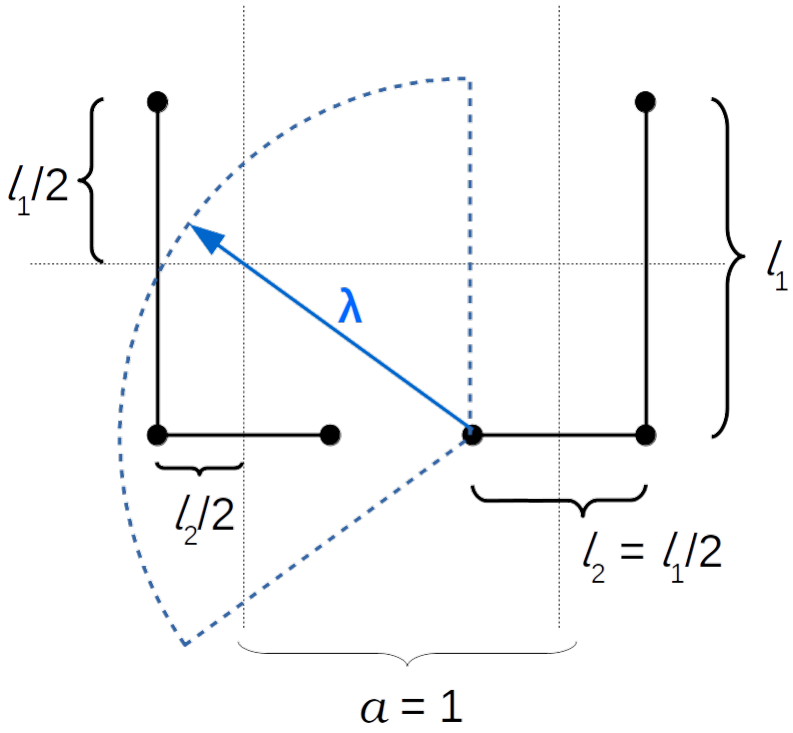


Figura 5.1: Partículas en forma de L de distinta quiralidad. Se señalan las longitudes de los segmentos de las partículas, la de las aristas y el rango de interacción. Las aristas de la red bisectan las partículas.

donde σ es un estado del sistema; i, j son los índices de las partículas; p_i, q_j son los índices de los sitios atractivos de las partículas i, j; r_{p_i,q_j} es su distancia relativa y $U(r_{p_i,q_j})$ es su potencial de interacción, que en este caso es el pozo cuadrado. Para que una partícula no interactúe consigo misma, en la suma se impone que $j \neq i$.

En la Fig. 5.1 λ se muestra como una flecha y el arco de círculo muestra que un sitio de la partícula derecha alcanza a interactuar con dos de los tres sitios de la otra partícula.

Un modelo similar ya ha sido estudiado por Quintana et al. [16], presentado en la Fig. 5.2. Dicho modelo consiste en un sistema de partículas con forma de palo de hockey: dos segmentos de recta unidos con un ángulo de separación θ. Dichas partículas se pueden mover libremente e interactúan entre ellas a través de potenciales infinitamente repulsivos. La finalidad de este estudio fue hallar si había transiciones de fase inducidas por efectos meramente entrópicos. Esto ocurre cuando el único potencial intermolecular es el de núcleo duro [17].

El trabajo aquí presentado se puede considerar un complemento de [16], ya que se investigan los fenómenos que ocurren por efectos meramente energéticos; es decir, cuando no hay potenciales de núcleo duro mediando las interacciones de las partículas. Debido a la forma en la que están situadas, las partículas del modelo nunca colisionan, así que solo pueden interactuar mediante los

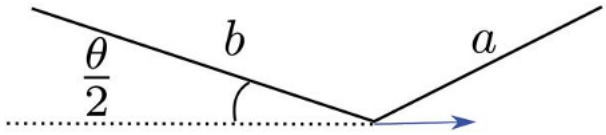


Figura 5.2: Partículas palo de hockey [16]

sitios atractivos. Dicho de otra manera: si una partícula se moviera hacia un sitio ya ocupado, el método de Montecarlo siempre rechazaría el movimiento debido al potencial de núcleo duro. Como las posiciones y orientaciones son limitadas en este modelo, en la práctica parecería que las partículas nunca colisionan.

5.2. Implementación

Se realizaron diversas simulaciones con el método de Montecarlo en los ensambles canónico y gran canónico utilizando el algoritmo de Metropolis, además de utilizar descenso gradual de temperatura. El algoritmo Metropolis Monte Carlo se instrumentó de la siguiente manera: el movimiento de Montecarlo de prueba equivale a un cambio de quiralidad de la particula seleccionada. El cambio se obtiene de dos maneras diferentes:

1. La partícula se refleja con respecto a la arista vertical de la red que bisecta el lado corto. Si, por ejemplo, el lado largo se encontraba a la derecha de la arista, ahora se encontrará a la izquierda de ésta y a la misma distancia que antes. El lado corto permanece inalterado. La Fig. 5.3 muestra las posiciones de los enantiómeros respecto al sitio de la red, marcado con una cruz.

2. La partícula se refleja con respecto a la arista horizontal de la red que bisecta el lado largo. Si, por ejemplo, el lado corto se encontraba abajo de la arista, ahora se encontrará arriba de ésta y a la misma distancia que antes. El lado largo permanece inalterado. La Fig. 5.4 muestra las posiciones de los enantiómeros respecto al sitio de la red, marcado con una cruz.

La aceptación del movimiento propuesto se realiza aplicando el algoritmo de Metropolis descrito en la Sección 4.1.

Como a priori no se sabe qué valores de T y λ generan separación quiral, se realizaron varias simulaciones para hallar las temperaturas y los rangos de interacción que generan alguna clase de separación quiral. La mayor parte de las simulaciones se realizaron para sistemas de tamaño $N = 50 \times 50$, aunque algunas se realizaron para $N = 100 \times 100$ y 200×200.

El método de descenso de temperatura se explica en la Sección 5.5.

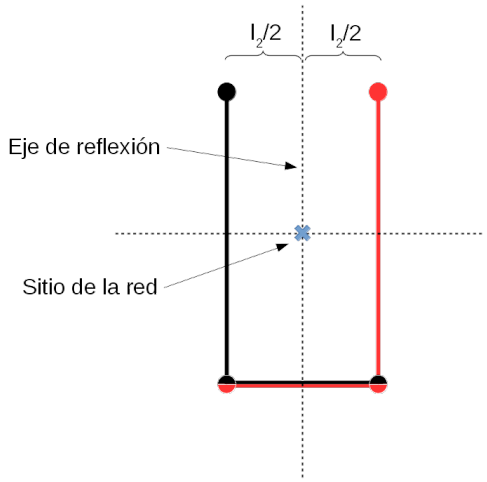


Figura 5.3: Un enantiómero reflejado por la arista vertical de la red genera el otro enantiómero. El lado corto de la molécula queda en la misma posición.

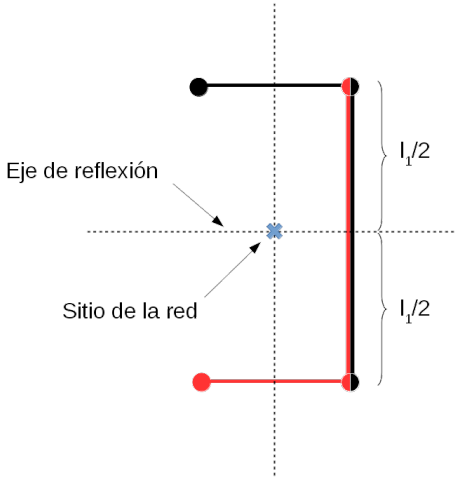


Figura 5.4: Un enantiómero reflejado por la arista horizontal de la red genera el otro enantiómero. El lado largo de la molécula queda en la misma posición.

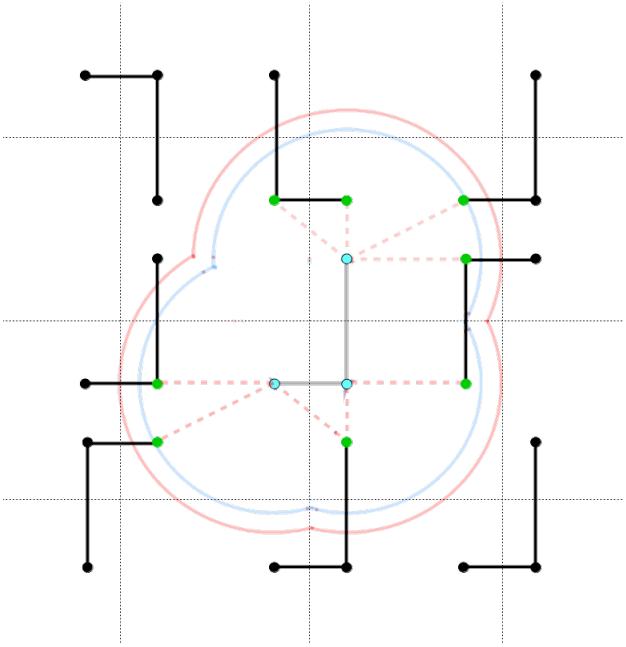
5.3. Elección de valores de λ

Una parte muy importante sobre la implementación es la manera en la que se escogieron los valores de λ para las simulaciones. Hay que tomar en cuenta que las posiciones que pueden ocupar las partículas (y sus sitios de interacción) son discretas, no continuas. Por esto, a pesar de que el valor del rango de interacción λ se puede escoger dentro de un conjunto continuo de valores, la naturaleza discreta de las posiciones de las partículas hace que los efectos en el sistema al variar λ cambien de manera discreta en vez de continua. Por ejemplo, si la distancia más corta entre dos sitios de interacción de partículas distintas es d_1 y se escoge una λ menor a d_1, las partículas no tendrán interacciones con sus vecinas. En ese sentido, para una misma configuración, cualquier valor de λ en el intervalo $[0, d_1)$ induce exactamente las mismas interacciones y por lo tanto, los mismos fenómenos. Si la segunda distancia más corta entre sitios de interacción es d_2, dos valores cualquiera de $\lambda \in [d_1, d_2)$ inducirán los mismos fenómenos, que pueden ser distintos a los de $\lambda \in [0, d_1)$.

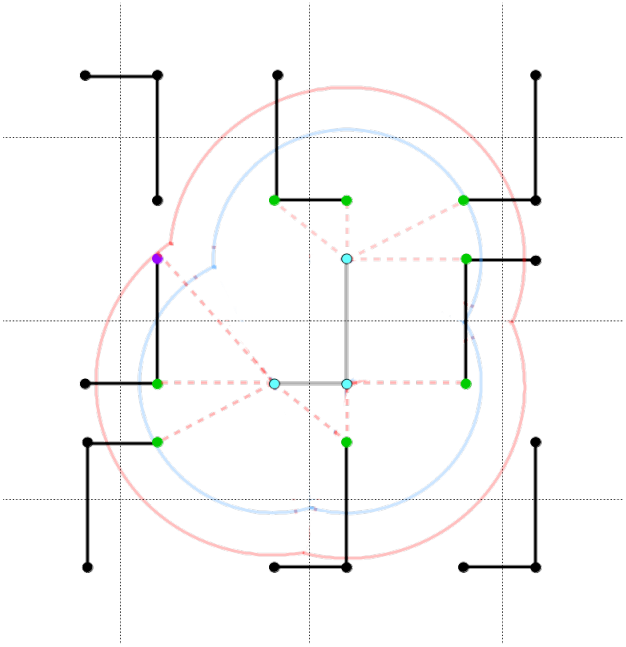
La Fig. 5.5 muestra cómo dos valores similares de λ pueden inducir las mismas o distintas interacciones con las partículas vecinas. En la primera imagen los segmentos de círculo denotan dos rangos de interacción centrados en los sitios de interacción, marcados con azul. Los sitios encerrados por ambos potenciales se señalan en verde. Como se puede ver en la figura, estos dos valores inducen las mismas interacciones, por lo que no es necesario realizar simulaciones para ambos. En la segunda imagen se puede ver que el rango más grande se ha aumentado un poco, induciendo una nueva interacción que podría generar fenómenos distintos en el sistema. Entonces es claro que para estudiar de manera eficiente el modelo es importante dividir los valores de λ en intervalos, delimitados por las posibles distancias entre sitios de interacción de partículas vecinas. Cada intervalo de λ induce nuevas interacciones, que pueden derivar en distintos fenómenos.

Para encontrar dichos rangos de λ, se calcularon todas las posibles distancias de un sitio de interacción a todos los sitios de interacción ubicados en los sitios de red vecinos. De la Fig. 5.3 se puede ver que, por la manera en la que se hace el cambio de quiralidad, los sitios atractivos de las partículas solo pueden ocupar cuatro posiciones alrededor de un sitio de red. Como cada sitio de red posee cuatro sitios vecinos laterales y otros cuatro vecinos laterales, en total hay ocho vecinos en los cuales puede situarse una partícula. Ya que hay ocho sitios de red vecinos y en cada uno los sitios atractivos de las partículas pueden ocupar solo cuatro posiciones, en total hay $8 \times 4 = 32$ lugares donde potencialmente se puede ubicar un sitio atractivo de una partícula vecina.

La Fig. 5.6 muestra todas las posibles posiciones de los sitios de interacción de la partícula central como puntos rojos y los sitios de las vecinas como puntos negros. También se muestra cómo se situarían partículas con una quiralidad en particular. Las partículas vecinas de la partícula central están numeradas del 1 al 8. Los posibles lugares donde puede ubicarse un sitio de interacción están numerados del 1 al 4. La Tabla 5.1 muestra el cálculo de las distancias a las que se encuentran dichos sitios de un sitio en particular. Esta Tabla se obtuvo utilizando SymPy, un paquete de Python para realizar computación simbólica. De esta manera se evitó calcular las 32 distancias a mano y sus valores numéricos. Varias de las 32 posibles distancias son numéricamente iguales, así que el número total de distancias distintas se redujo a 20.



(a) Dos rangos de interacción que inducen las mismas interacciones.



(b) Dos rangos de interacción que inducen distintas interacciones. El rango mostrado en azul es igual que el de la imagen de arriba, mientras que el rojo se ha aumentado. Además, el rango mostrado en rojo induce una interacción adicional con el sitio mostrado en púrpura.

Figura 5.5: Efectos de la longitud del rango de interacción sobre el número de interacciones. Los rangos de interacción mostrados en rojo y azul inducen interacciones con los sitios mostrados en verde.

5.4. Energía de interacción de las partículas

Con los valores calculados de λ ya se pueden obtener las energías de las partículas y, consecuentemente, de los sistemas a estudiar. La energía de un solo sitio de interacción de una partícula es, de manera práctica, el número de sitios atractivos de otras partículas que se encuentran a una distancia menor o igual a λ. La energía de una partícula es la suma de las energías de sus sitios atractivos. Si el sitio atractivo de otra partícula se encuentra dentro del rango de interacción de más de un sitio atractivo de la partícula a estudiar, contribuirá más de una vez a la energía total. Las partículas pueden interactuar con cualquier otra partícula que se encuentre dentro del rango de interacción, no solamente con sus primeras vecinas.

Un rango de interacción especial es $\lambda = 1$ ya que es la distancia entre dos sitios de red vecinos y, si una partícula y su vecina poseen la misma quiralidad, este es el rango mínimo con el cual cada sitio de interacción de una partícula puede interactuar con el sitio equivalente en la otra partícula. Por esta razón se propone que al menos para $\lambda = 1$ se puede generar separación quiral a bajas temperaturas. En la Fig. 5.7 se muestran dichas interacciones en rojo. Además están las interacciones de los sitios atractivos que se encuentra a $d < 1$ entre sí. Éstas se muestran en azul.

Si L es el lado de un sistema, $N = L^2$ es el número de partículas. Cada simulación consistió en L^5 pasos, ya que se observó que en ese orden de número de pasos los sistemas se equilibraban. Para un sistema de 50×50, las simulaciones consistieron en $(50^5 = 3.125 \times 10^8$ pasos de Montecarlo.

El muestreo de sistema se hizo cada N pasos, lo cual constituye un barrido del sistema (definido en el Apéndice A). Por lo tanto, las simulaciones de un sistema de tamaño 50×50 consistieron en $50^3 = 125000 = 1.25 \times 10^5$ muestreos del sistema cada 2500 pasos.

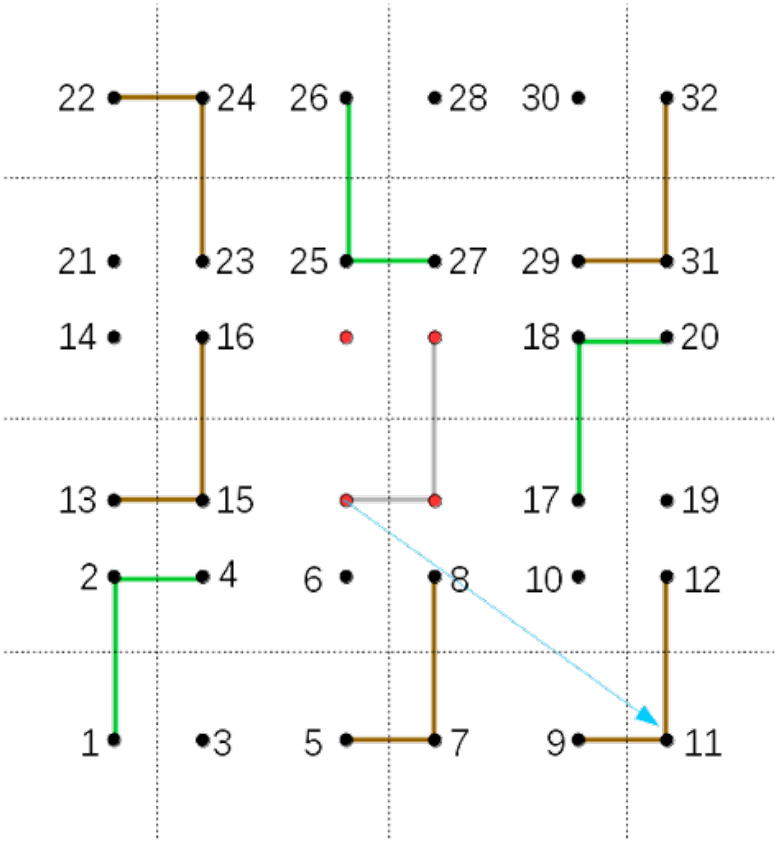


Figura 5.6: Posibles distancias entre un sitio de interacción y sus vecinos

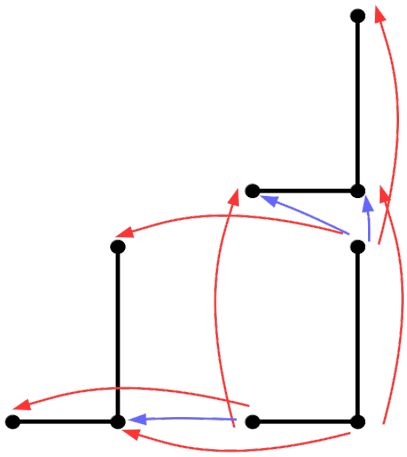


Figura 5.7: Con $\lambda = 1$, cada sitio interactúa con su equivalente (rojo) además de que interactúa con las más cercanas (azul)

Índice(s)	Distancia d	d cuando $l_1 = 0.9$
6	$1 - l_1$	0.10000
8	$\sqrt{l_2^2 + (l_1 - 1)^2}$	0.46098
15	$1 - l_2$	0.55000
4	$\sqrt{(l_1 - 1)^2 + (l_2 - 1)^2}$	0.55902
5, 13, 17, 25	1	1.00000
2, 10	$\sqrt{(l_1 - 1)^2 + 1}$	1.00499
16	$\sqrt{l_1^2 + (l_2 - 1)^2}$	1.05476
7, 27	$\sqrt{l_2^2 + 1}$	1.09659
3, 23	$\sqrt{(l_2 - 1)^2 + 1}$	1.14128
14, 18	$\sqrt{l_1^2 + 1}$	1.34537
1, 9, 21, 29	$\sqrt{2}$	1.41422
19	$1 + l_2$	1.45000
12	$\sqrt{(l_1 - 1)^2 + (l_2 + 1)^2}$	1.45345
20	$\sqrt{l_1^2 + (l_2 + 1)^2}$	1.70661
11, 31	$\sqrt{(l_2 + 1)^2 + 1}$	1.76140
26	$1 + l_1$	1.90000
28	$\sqrt{l_2^2 + (l_1 + 1)^2}$	1.95257
24	$\sqrt{l_1^2 + 1}$	1.97801
22, 30	$\sqrt{(l_1 + 1)^2 + 1}$	2.14710
32	$\sqrt{l_1^2 + (l_2 + 1)^2}$	2.39009

Tabla 5.1: Separación entre sitios de interacción de partículas vecinas y valores de λ para las simulaciones, de menor a mayor. Los índices son los indicados en la Fig. 5.6.

5.5. Elección de temperaturas

5.5.1. Descenso gradual de temperatura

En la Sección 5.2 se mencionó que se aplicó un descenso gradual de temperatura a cada simulación. Esto fue así para que cada sistema alcanzara el equilibrio de manera más sencilla. En las primeras simulaciones que se hicieron, sin descenso de temperatura, se observó que algunos estados no lograban alcanzar los estados de equilibrio, por lo que se decidió instrumentar este procedimiento.

El procedimiento es el siguiente: se empieza desde una temperatura alta T_h (en alguna que se hubiera visto muy poco orden en los estados finales) y se desciende gradualmente hasta una temperatura deseada T_l. La forma más sencilla es descender de manera lineal: si se quiere realizar la simulación a S temperaturas distintas, se divide el intervalo $[T_h, T_l]$ en $S-1$ subintervalos del mismo tamaño. Cada subintervalo mide $dT = \dfrac{T_h - T_l}{S-1}$, por lo que en total hay S temperaturas distintas $T_i, i = 1, ..., S$, donde $T_1 = T_h$, $T_i = T_{i-1} - dT$ y $T_S = T_l$. Después se calcula cada cuántos pasos de Montecarlo se bajará la temperatura: si en la simulación se realizan M pasos de Montecarlo, se realiza un descenso de temperatura cada $\dfrac{M}{S}$ pasos. Cada simulación empieza a $T = T_h$ y después de $\dfrac{M}{S}$, se desciende a $T = T_h - dT$. Esto se aplica sucesivamente para descender la temperatura (es decir, cada $\dfrac{i * M}{S}$ pasos se fija la temperatura en T_i) hasta que los últimos $\dfrac{M}{S}$ pasos de la simulación se realizan a $T = T_l$.

Para estas simulaciones, sin embargo, se decidió descender la temperatura de manera cuadrática. Se observó que las energía descendían muy poco al principio y abruptamente después. Con un descenso cuadrático, el descenso fue un poco más uniforme. El descenso lineal se puede transformar en un descenso cuadrático si, para cada T_i obtenida de manera lineal, se le aplica la transformación

$$T_i' = aT_i^2 + b,$$

donde a y b se obtienen si se aplican condiciones para que la transformación deje inalteradas a T_h y a T_l; es decir, para que

$$T_h' = T_h = aT_h^2 + b$$

$$T_l' = T_l = aT_l^2 + b$$

Lo anterior constituye un sistema de ecuaciones para a y b, cuya solución es

$$a = \frac{T_l - T_h}{T_l^2 - T_h^2}$$

$$b = T_l - \left(\frac{T_l - T_h}{T_l^2 - T_h^2}\right) T_l^2$$

5.5.2. Temperaturas finales

Las temperaturas finales T_l de las simulaciones fueron de $T = 0.1$ a $T = 1.0$ en intervalos de $T = 0.1$ y de $T = 1.0$ a $T = 2.0$ en intervalos de $T = 0.2$. El razonamiento para encontrar el rango adecuado de temperaturas es el siguiente: en el modelo de Ising, los estados desordenados ocurren

a temperaturas reducidas T mayores a $T_c \approx 2.3$ y los ordenados a temperaturas menores. El mayor cambio posible del valor absoluto de la energía al invertir un espín es $|\Delta U| = 8$. Con este rango de valores de T y $|\Delta U|$ se generan los estados ordenados del modelo de Ising a partir del peso del Boltzmann. Es sencillo darse cuenta de que $|\Delta U| \lesssim 10$ en el modelo de partículas L al cambiar la quiralidad de una partícula, al menos para valores de λ cercanos a 1, por lo que las temperaturas a las que se podría encontrar separación quiral estarían alrededor de $T \sim 1$. Con esta información se propusieron las temperaturas arriba mencionadas y con las que se hallaron diversos resultados.

Con parámetros similares se realizaron simulaciones en el ensamble gran canónico. En este caso también se inició con un número igual de enantiómeros, pero solamente ocupaban el 10 % del sistema. El potencial químico se fijó en $\mu = 40$, para que el sistema se fuera poblando de partículas. Este valor se escogió por prueba y error. Como el límite termodinámico de todos los ensambles estadísticos es el mismo, se espera que ambos tipos de simulaciones generen resultados congruentes. De hecho, los resultados de estas simulaciones son los mismos para las simulaciones en los ensambles canónico y gran canónico.

El lenguaje utilizado para realizar las simulaciones fue FORTRAN90. Los cálculo del exceso enantiomérico, del número y tamaño de cúmulos y las gráficas fueron hechos en Julia, en parte con las dependencias Gadfly y ColorBrewer.

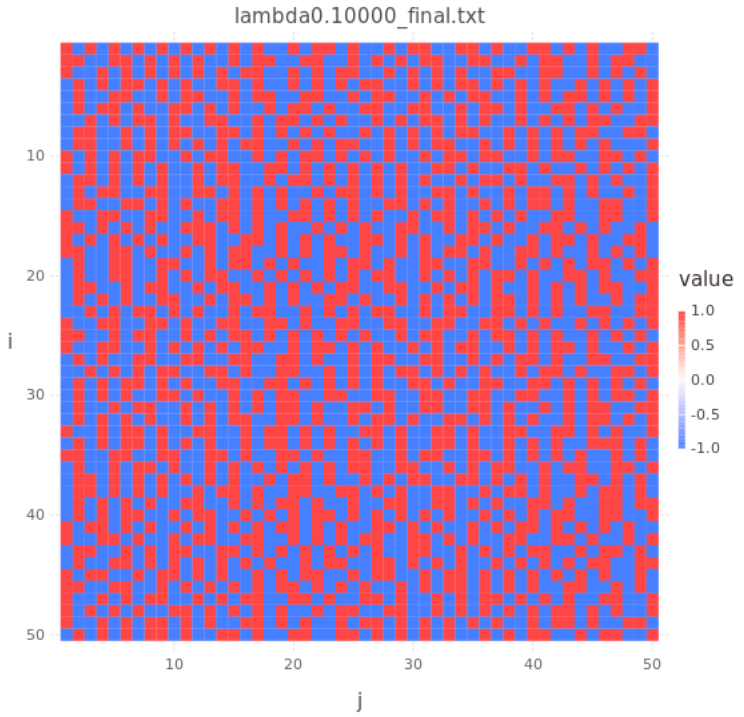
Capítulo 6

Resultados

Las simulaciones realizadas con el método de Montecarlo en los ensambles canónico y gran canónico generan resultados congruentes entre sí. Las siguientes Figuras se obtuvieron con un tamaño de 50×50, a $T_l = 0.1$ con el ensamble canónico pero las obtenidas con el ensamble gran canónico se ven esencialmente iguales. Los sistemas obtienen básicamente las mismas configuraciones y energías por partícula con ambos ensambles. Lo mismo ocurre al variar el tamaño del sistema: se obtienen esencialmente las mismas configuraciones y energías por partícula.

(Nota: a partir de aquí se denotará T_l simplemente como T). Las figuras muestran que en el rango de temperaturas $T = [0.1, 0.3]$ se halló que se forman patrones en los sistemas. En los sistemas con $\lambda = 1.0, \sqrt{2} \approx 1.41421$ se observaron estados casi enantiopuros. En los demás parece que no, por lo que se estudiaron más detalladamente.

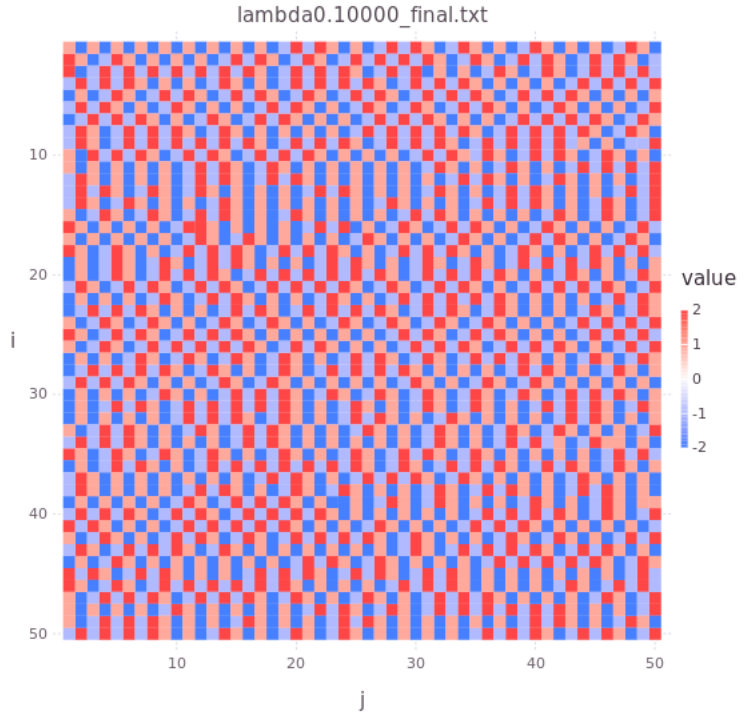
A partir de $T = 0.4$ se empieza a perder orden en los sistemas con λ pequeña. Para $\lambda = 2.39008$ la pérdida de orden empieza a ser importante a partir de $T = 1.4$.



Azul :L o ⅂ (izquierdas)

Rojo: ⅃ o ⌜ (derechas)

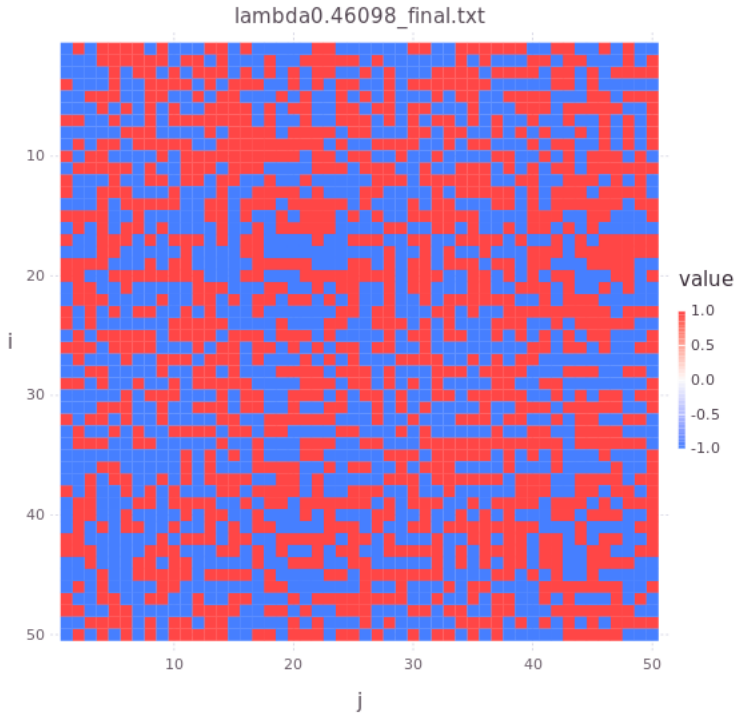
(a) Representación de la quiralidad del sistema con $\lambda = 0.1$.



Azul claro: L (izquierda)
Azul oscuro: ⅂ (izquierda)
Rojo claro: ⅃ (derecha)
Rojo oscuro: ⌜ (derecha)

(b) Representación de la quiralidad y orientación del sistema con $\lambda = 0.1$.

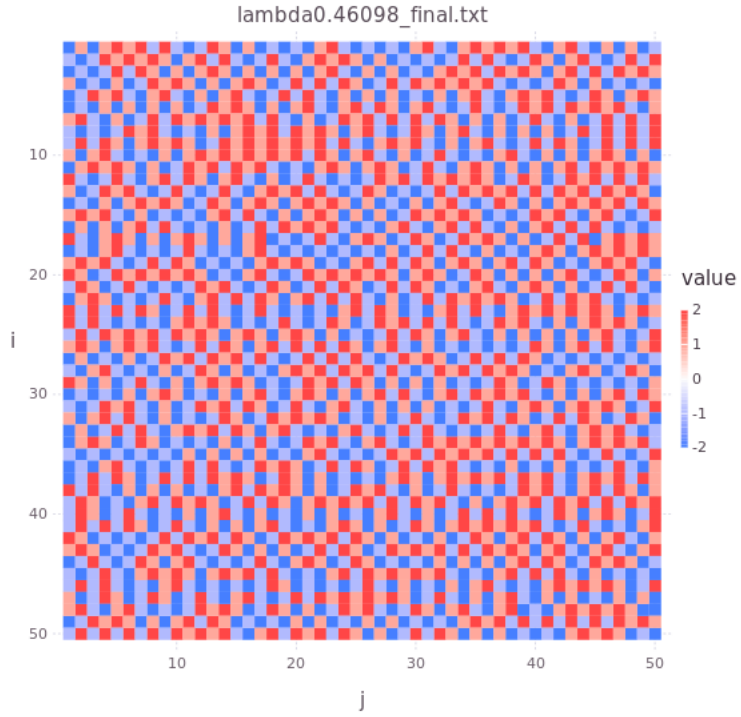
Figura 6.1: Sistema con $\lambda = 0.1$.



Azul :L o ⅂ (izquierdas)

Rojo: ⅃ o ⅂ (derechas)

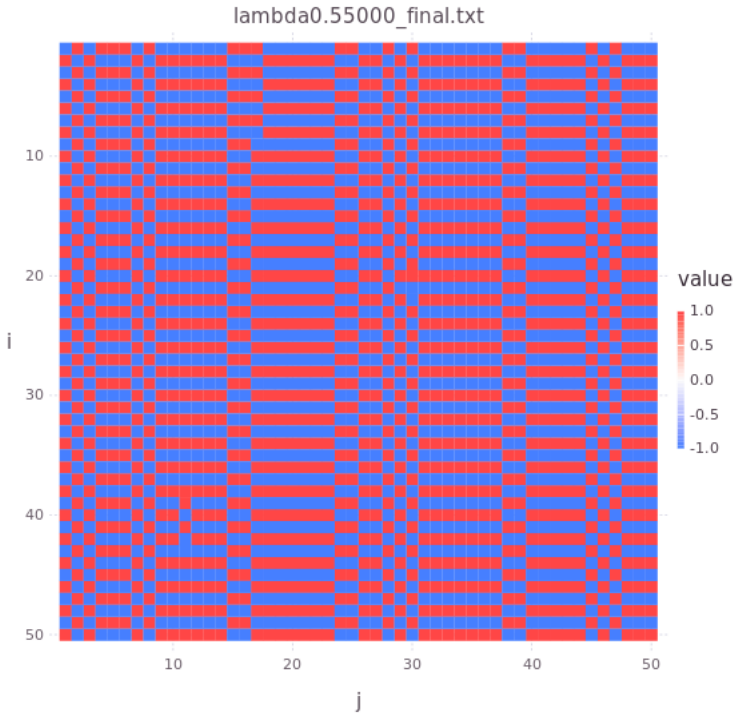
(a) $\lambda = 0.46098$ con quiralidad



Azul claro: L (izquierda)
Azul oscuro: ⅂ (izquierda)
Rojo claro: ⅃ (derecha)
Rojo oscuro: ⅂ (derecha)

(b) $\lambda = 0.46098$ con quiralidad y orientación

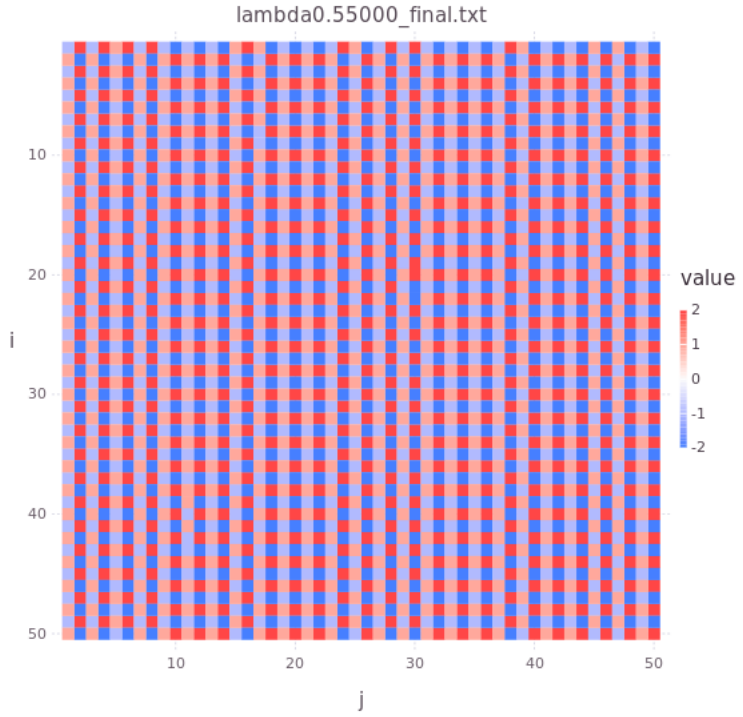
Figura 6.2: Sistema con $\lambda = 0.46098$.



Azul :L o ⅂ (izquierdas)

Rojo: ⅃ o ⌜ (derechas)

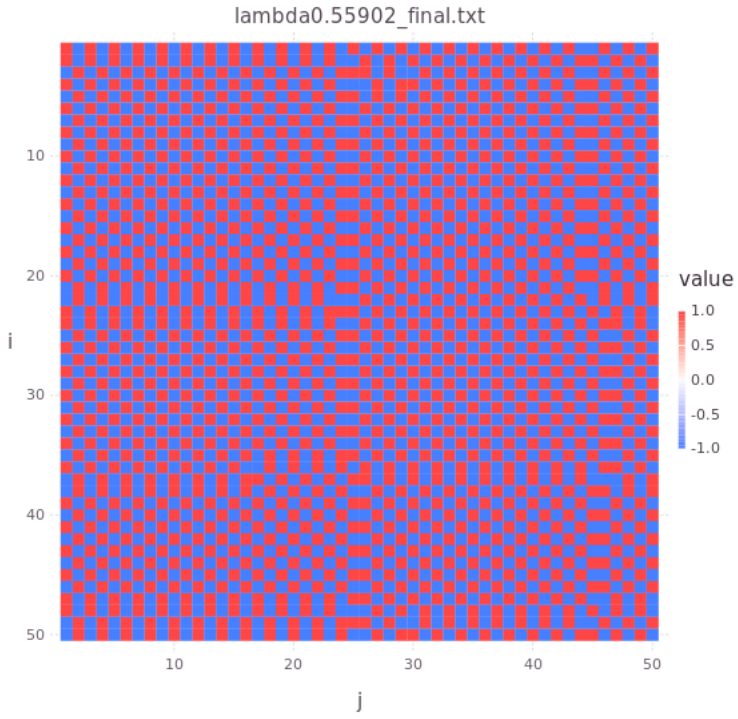
(a) $\lambda = 0.55$ con quiralidad



Azul claro: L (izquierda)
Azul oscuro: ⅂ (izquierda)
Rojo claro: ⅃ (derecha)
Rojo oscuro: ⌜ (derecha)

(b) $\lambda = 0.55$ con quiralidad y orientación

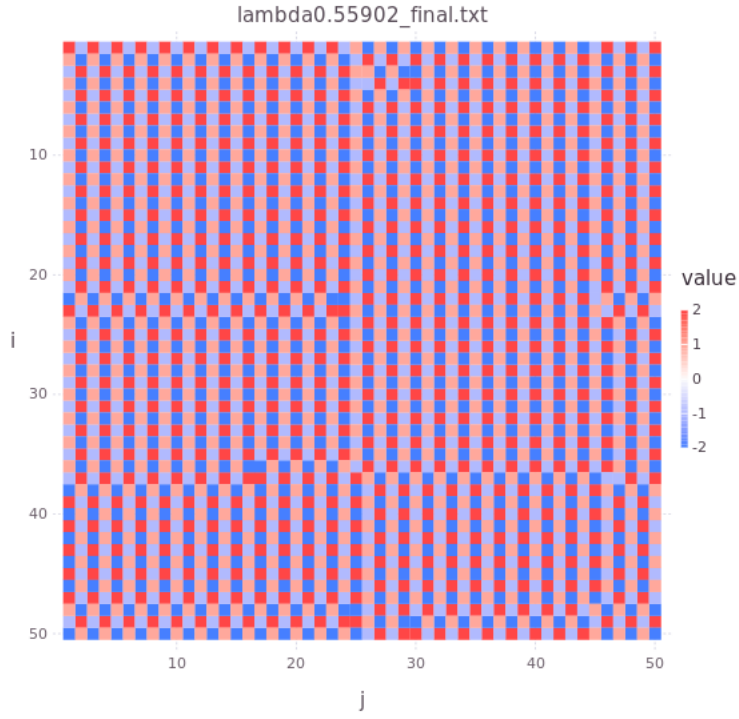
Figura 6.3: Sistema con $\lambda = 0.55$.



Azul :L o ⅂ (izquierdas)

Rojo: ⅃ o ⅂ (derechas)

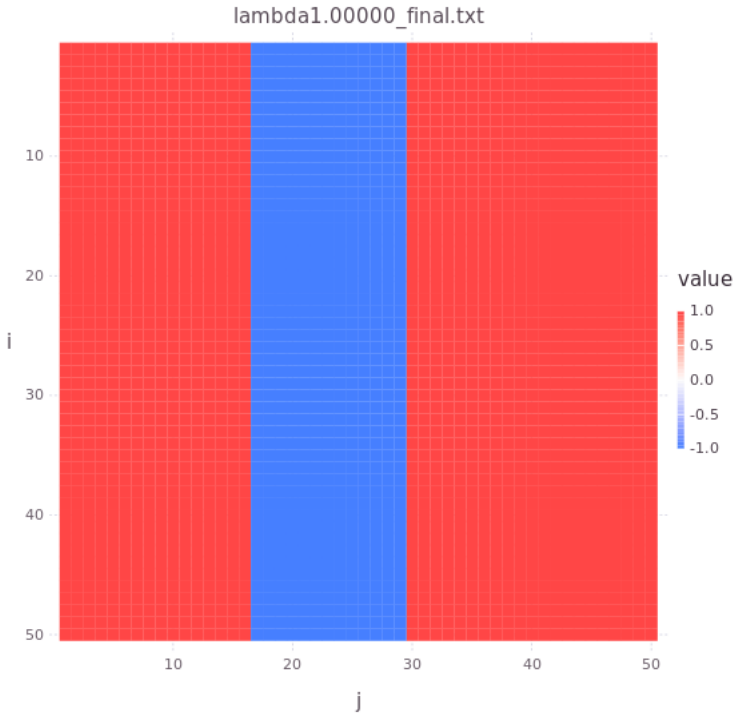
(a) $\lambda = 0.55902$ con quiralidad



Azul claro: L (izquierda)
Azul oscuro: ⅂ (izquierda)
Rojo claro: ⅃ (derecha)
Rojo oscuro: ⅂ (derecha)

(b) $\lambda = 0.55902$ con quiralidad y orientación

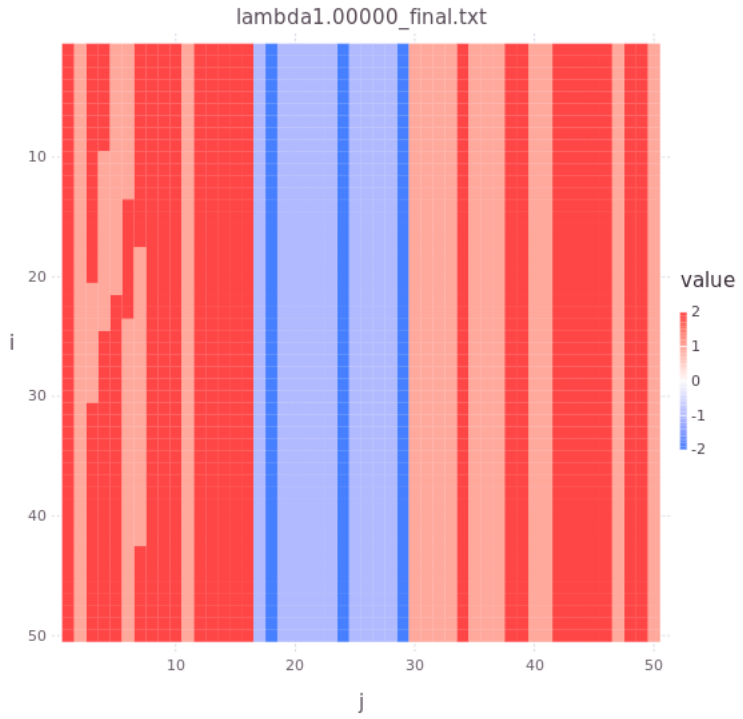
Figura 6.4: Sistema con $\lambda = 0.55902$.



Azul :L o ⅂ (izquierdas)

Rojo: ⅃ o ⌜ (derechas)

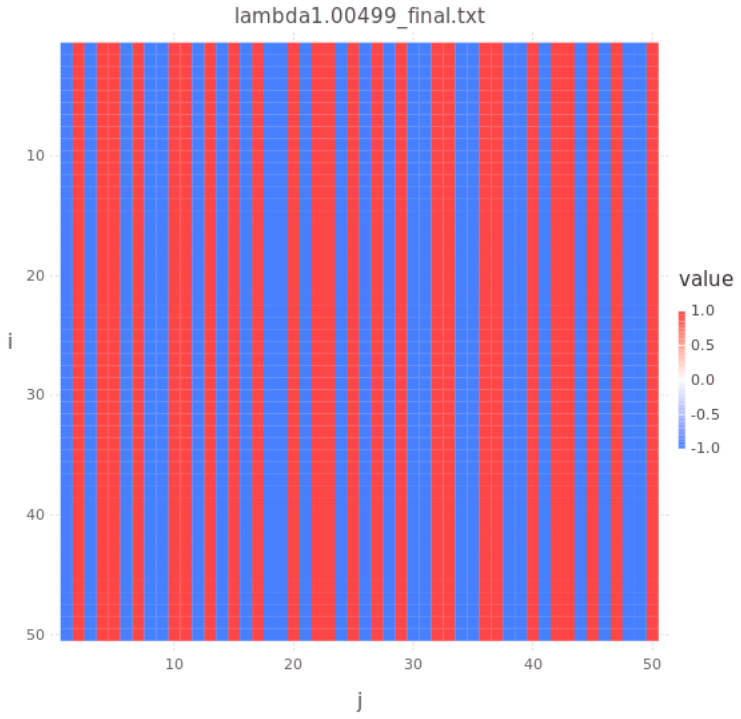
(a) $\lambda = 1.0$ con quiralidad



Azul claro: L (izquierda)
Azul oscuro: ⅂ (izquierda)
Rojo claro: ⅃ (derecha)
Rojo oscuro: ⌜ (derecha)

(b) $\lambda = 1.0$ con quiralidad y orientación

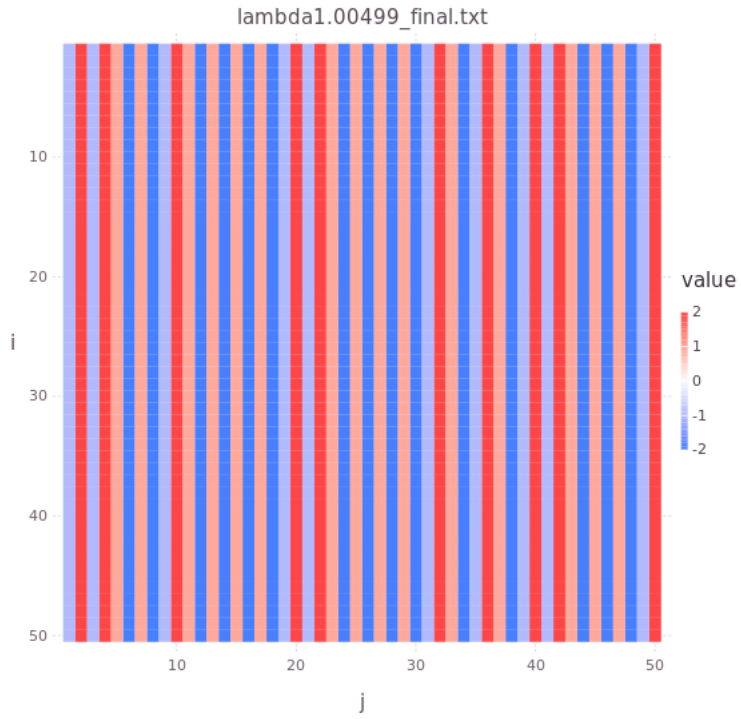
Figura 6.5: Sistema con $\lambda = 1.0$.



Azul :L o ⅂ (izquierdas)

Rojo: ⅃ o ⌜ (derechas)

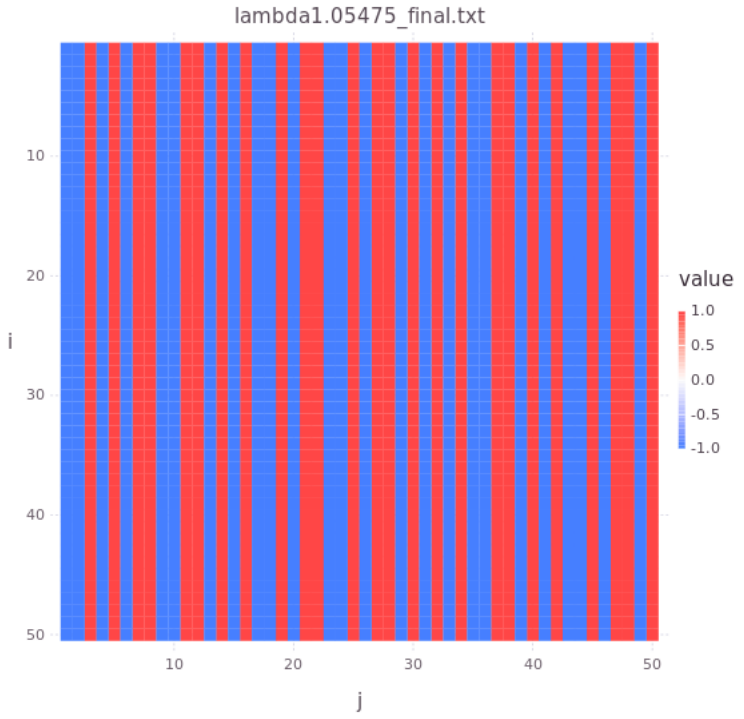
(a) $\lambda = 1.00499$ con quiralidad



Azul claro: L (izquierda)
Azul oscuro: ⅂ (izquierda)
Rojo claro: ⅃ (derecha)
Rojo oscuro: ⌜ (derecha)

(b) $\lambda = 1.00499$ con quiralidad y orientación

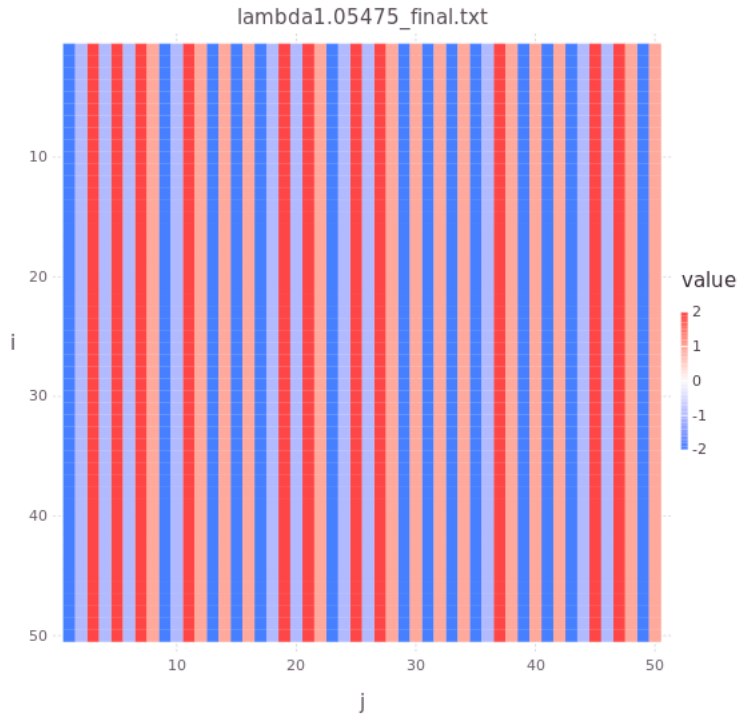
Figura 6.6: Sistema con $\lambda = 1.00499$.



Azul :L o ⅂ (izquierdas)

Rojo: ⅃ o ⌜ (derechas)

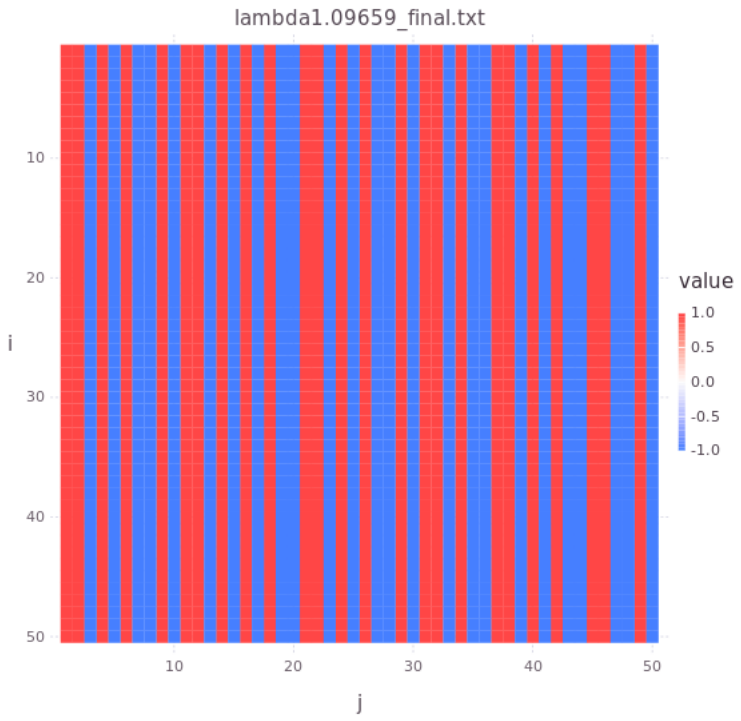
(a) $\lambda = 1.05475$ con quiralidad



Azul claro: L (izquierda)
Azul oscuro: ⅂ (izquierda)
Rojo claro: ⅃ (derecha)
Rojo oscuro: ⌜ (derecha)

(b) $\lambda = 1.05475$ con quiralidad y orientación

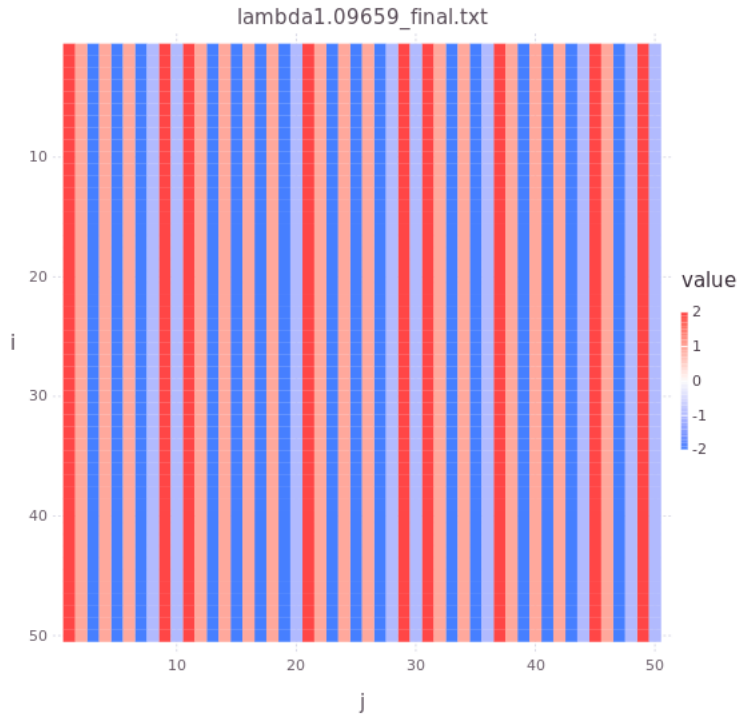
Figura 6.7: Sistema con $\lambda = 1.05475$.



Azul :L o ⅂ (izquierdas)

Rojo: ⅃ o ⅂ (derechas)

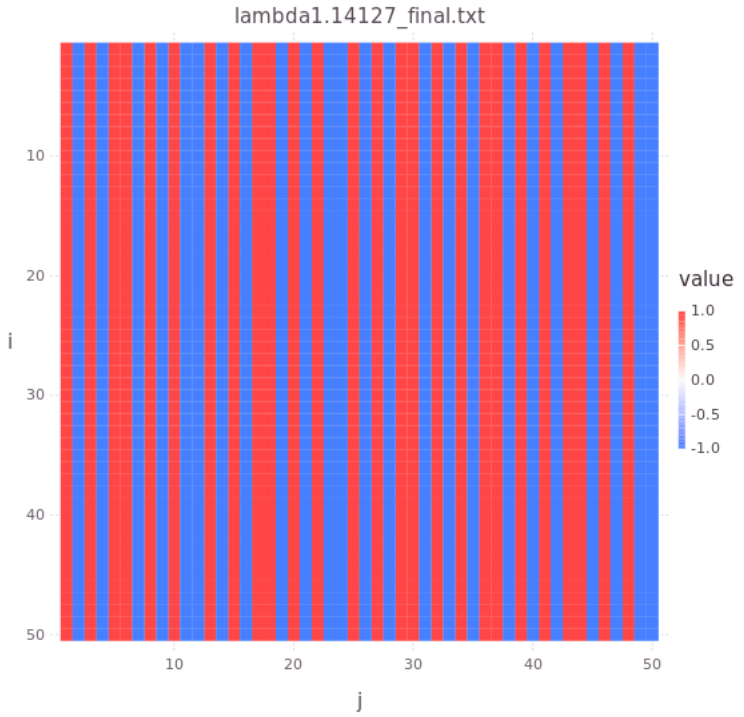
(a) $\lambda = 1.09659$ con quiralidad



Azul claro: L (izquierda)
Azul oscuro: ⅂ (izquierda)
Rojo claro: ⅃ (derecha)
Rojo oscuro: ⅂ (derecha)

(b) $\lambda = 1.09659$ con quiralidad y orientación

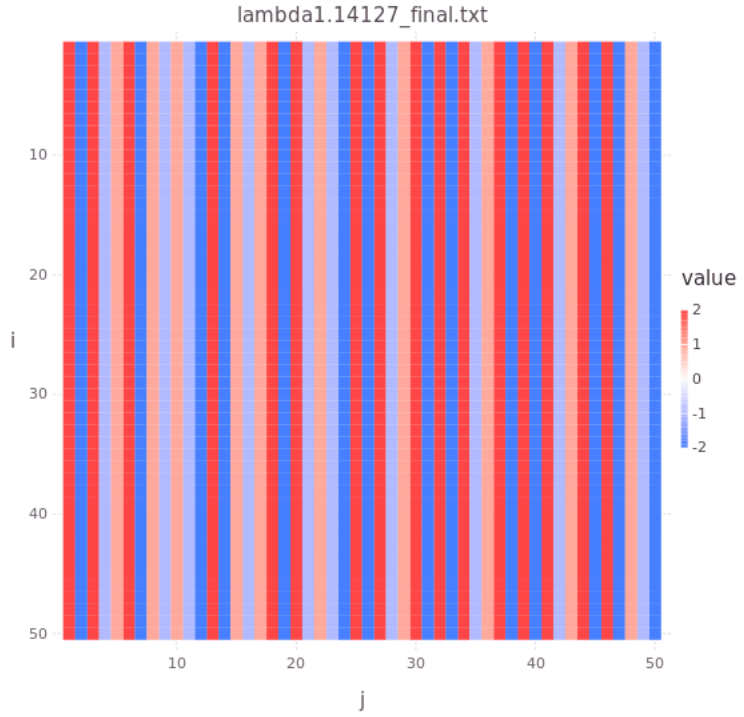
Figura 6.8: Sistema con $\lambda = 1.09659$.



Azul :L o ⅂ (izquierdas)

Rojo: ⅃ o ⌜ (derechas)

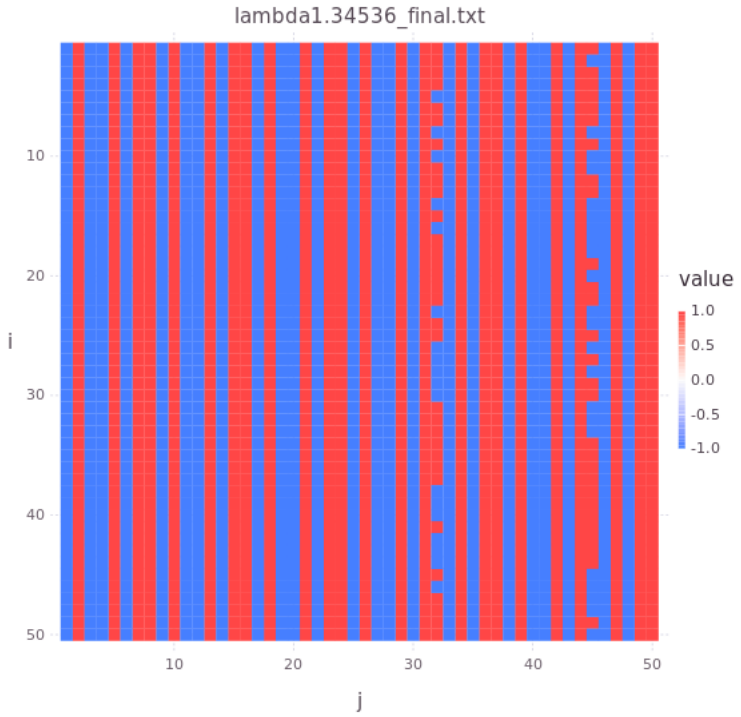
(a) $\lambda = 1.14127$ con quiralidad



Azul claro: L (izquierda)
Azul oscuro: ⅂ (izquierda)
Rojo claro: ⅃ (derecha)
Rojo oscuro: ⌜ (derecha)

(b) $\lambda = 1.14127$ con quiralidad y orientación

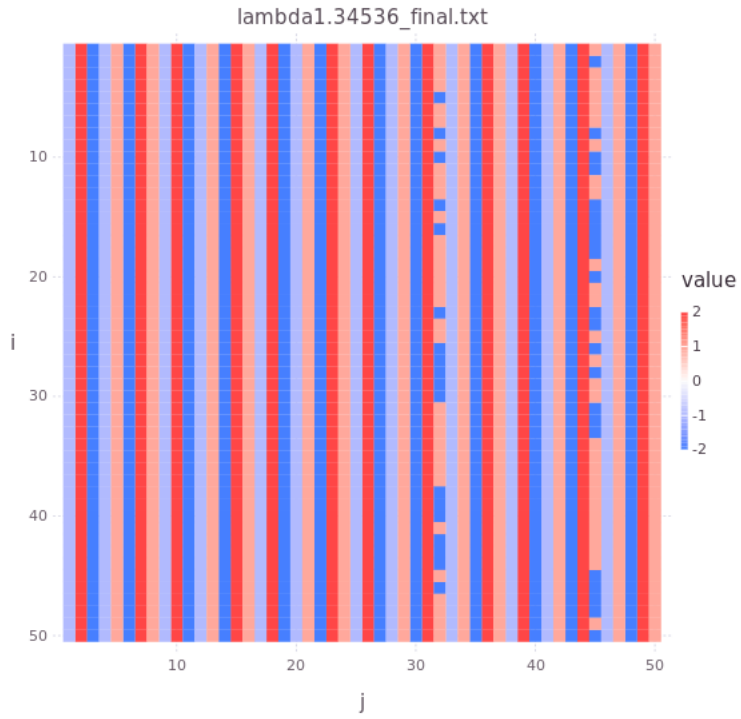
Figura 6.9: Sistema con $\lambda = 1.14127$.



Azul :L o ⅂ (izquierdas)

Rojo: ⅃ o ⌜ (derechas)

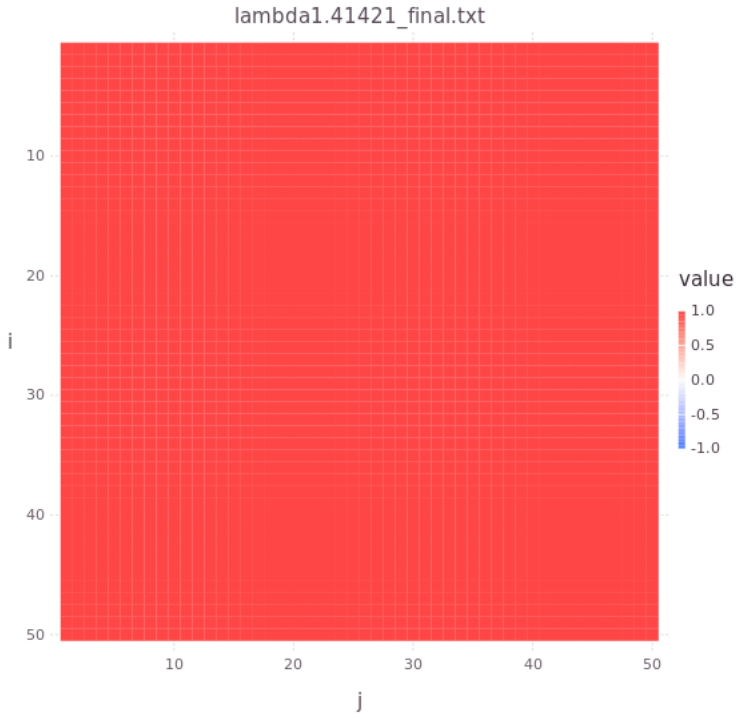
(a) $\lambda = 1.34536$ con quiralidad



Azul claro: L (izquierda)
Azul oscuro: ⅂ (izquierda)
Rojo claro: ⅃ (derecha)
Rojo oscuro: ⌜ (derecha)

(b) $\lambda = 1.34536$ con quiralidad y orientación

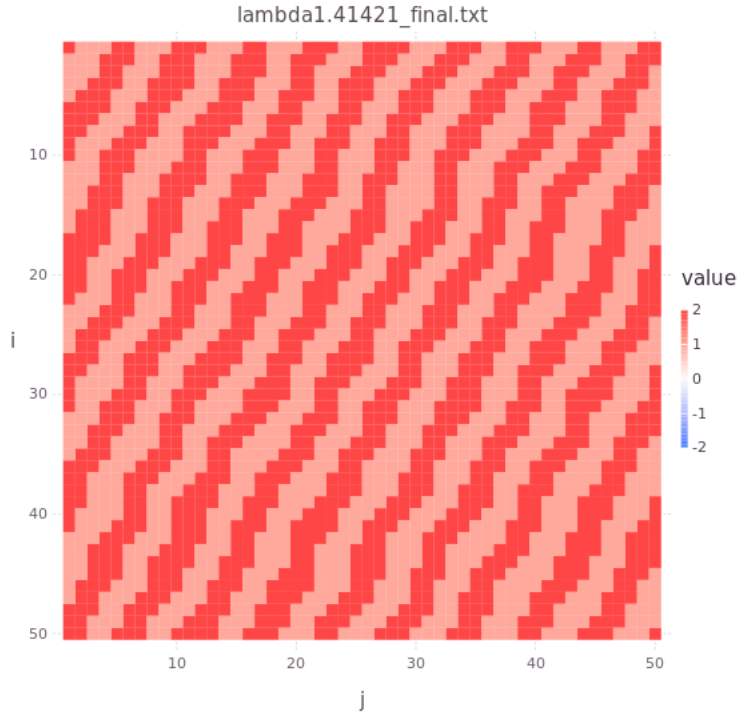
Figura 6.10: Sistema con $\lambda = 1.34536$.



Azul :L o ⅂ (izquierdas)

Rojo: ⅃ o ⌜ (derechas)

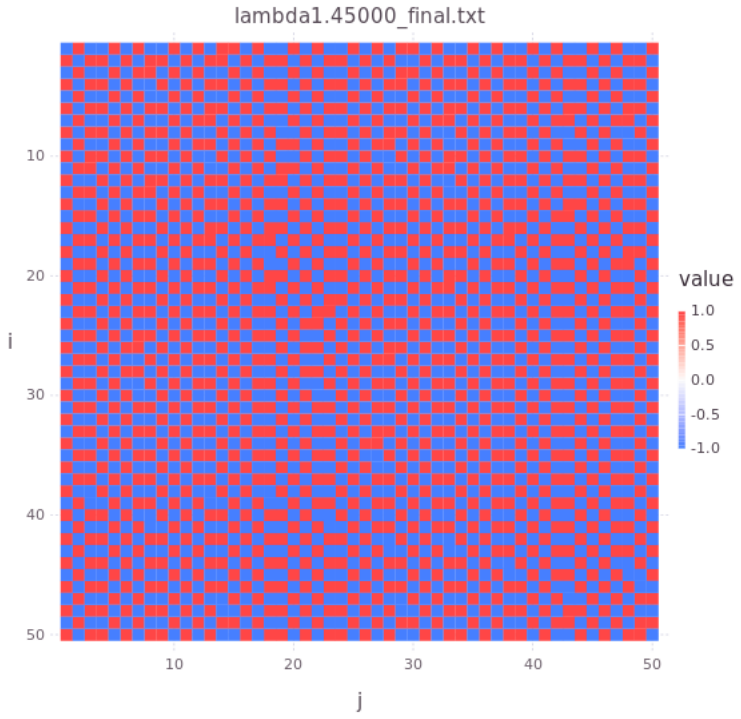
(a) $\lambda = 1.41421$ con quiralidad



Azul claro: L (izquierda)
Azul oscuro: ⅂ (izquierda)
Rojo claro: ⅃ (derecha)
Rojo oscuro: ⌜ (derecha)

(b) $\lambda = 1.41421$ con quiralidad y orientación

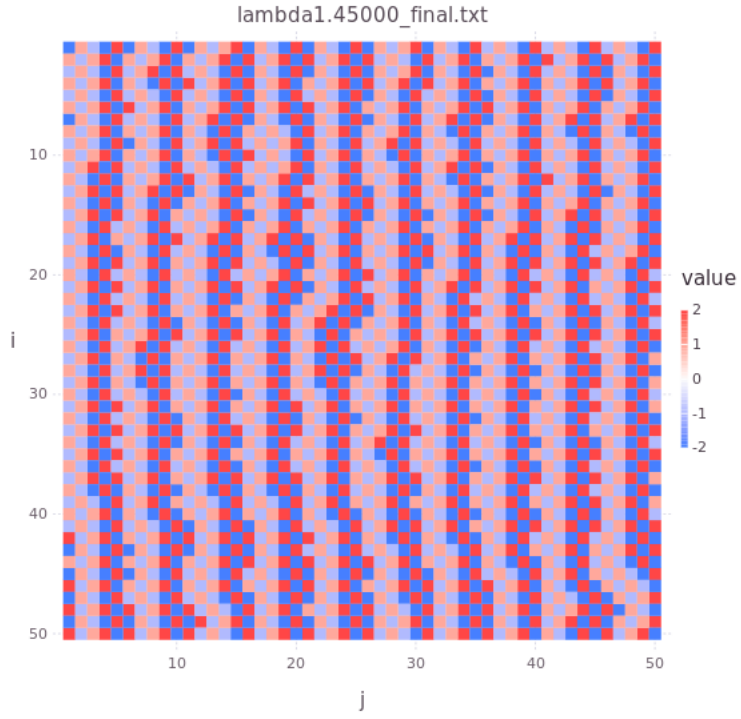
Figura 6.11: Sistema con $\lambda = 1.41421$.



Azul :L o ⅂ (izquierdas)

Rojo: ⅃ o ⅂ (derechas)

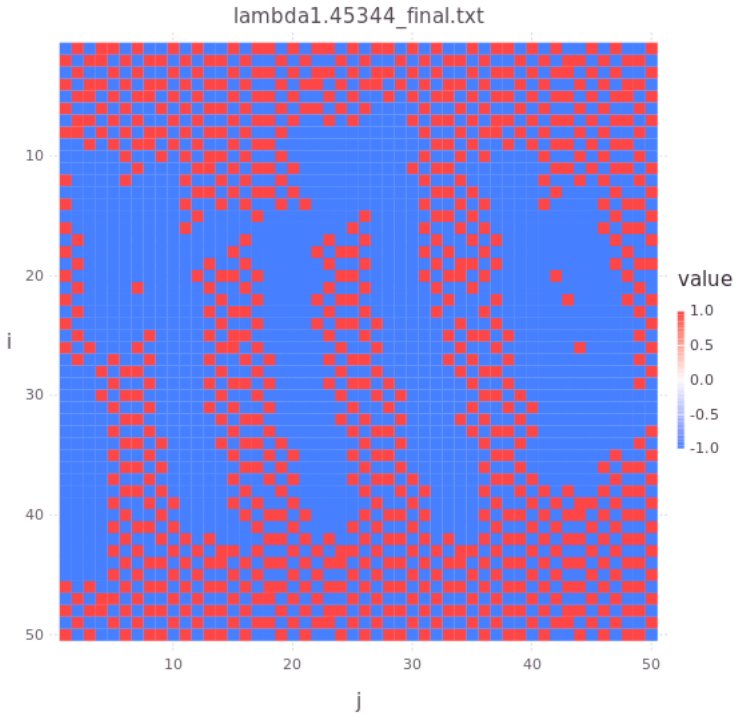
(a) $\lambda = 1.45$ con quiralidad



Azul claro: L (izquierda)
Azul oscuro: ⅂ (izquierda)
Rojo claro: ⅃ (derecha)
Rojo oscuro: ⅂ (derecha)

(b) $\lambda = 1.45$ con quiralidad y orientación

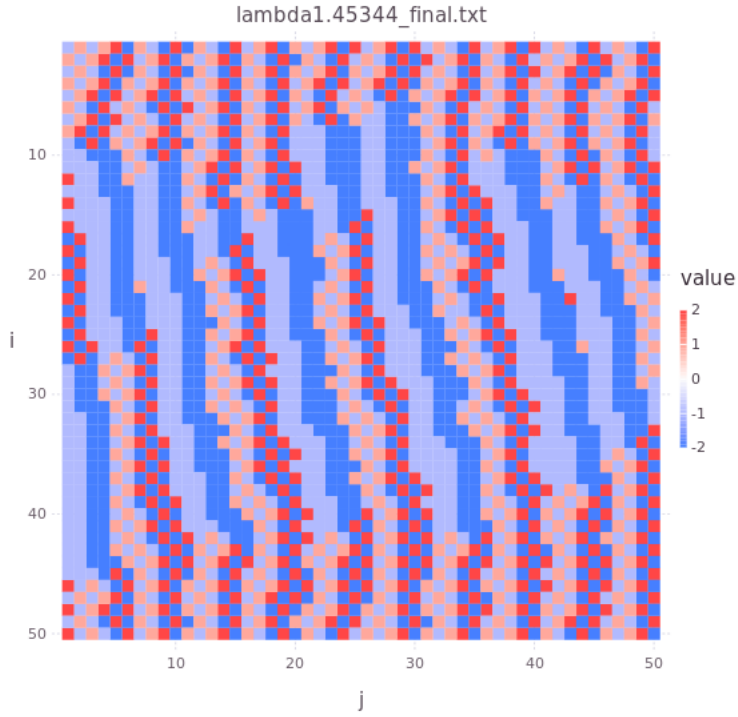
Figura 6.12: Sistema con $\lambda = 1.45$.



Azul :L o ⅂ (izquierdas)

Rojo: ⅃ o ⌜ (derechas)

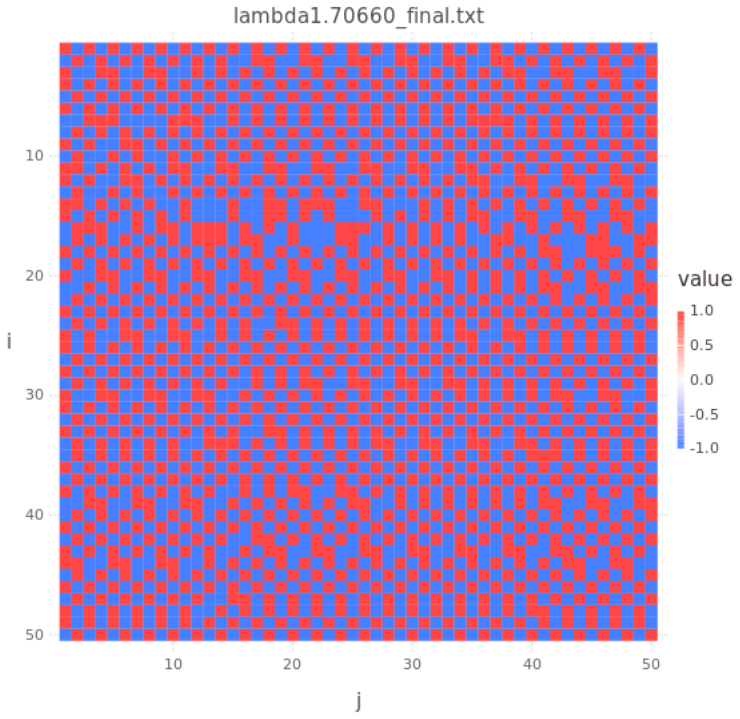
(a) $\lambda = 1.45344$ con quiralidad



Azul claro: L (izquierda)
Azul oscuro: ⅂ (izquierda)
Rojo claro: ⅃ (derecha)
Rojo oscuro: ⌜ (derecha)

(b) $\lambda = 1.45344$ con quiralidad y orientación

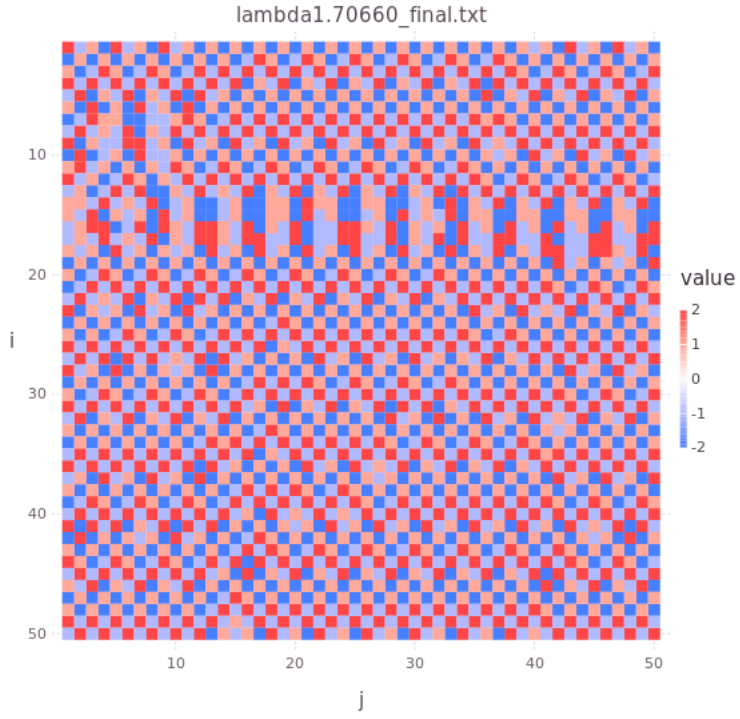
Figura 6.13: Sistema con $\lambda = 1.45344$.



Azul :L o ⅂ (izquierdas)

Rojo: ⅃ o ⅂ (derechas)

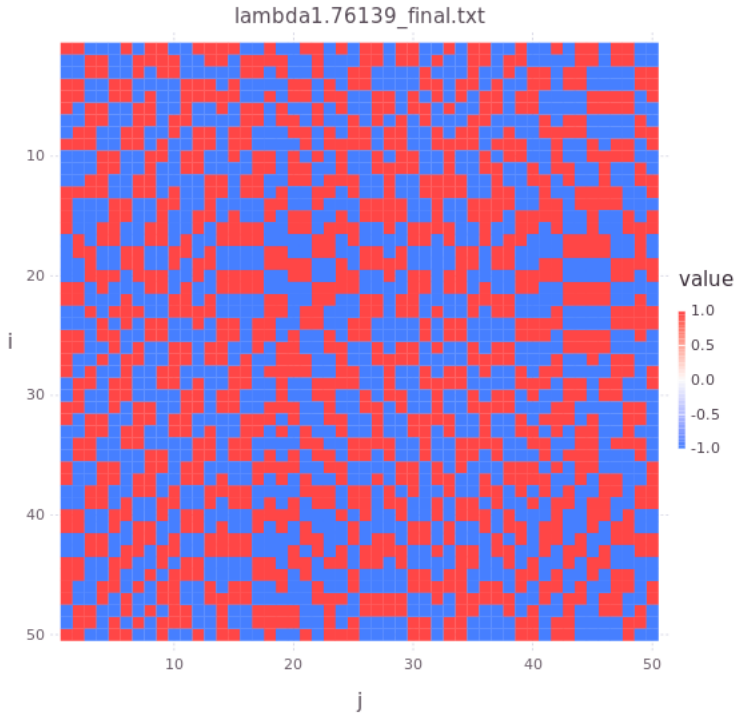
(a) $\lambda = 1.70660$ con quiralidad



Azul claro: L (izquierda)
Azul oscuro: ⅂ (izquierda)
Rojo claro: ⅃ (derecha)
Rojo oscuro: ⅂ (derecha)

(b) $\lambda = 1.70660$ con quiralidad y orientación

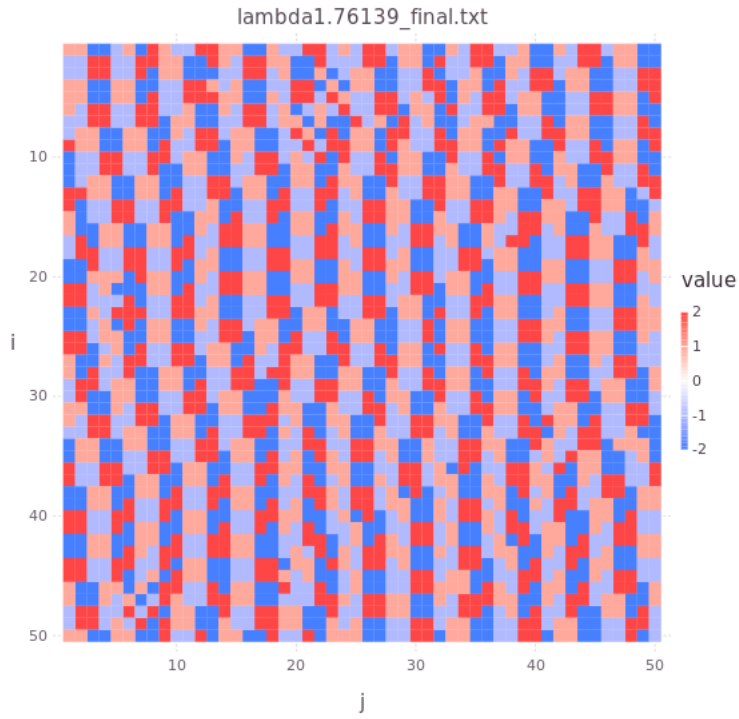
Figura 6.14: Sistema con $\lambda = 1.70660$.



Azul :⌞ o ⌝ (izquierdas)

Rojo: ⌟ o ⌜ (derechas)

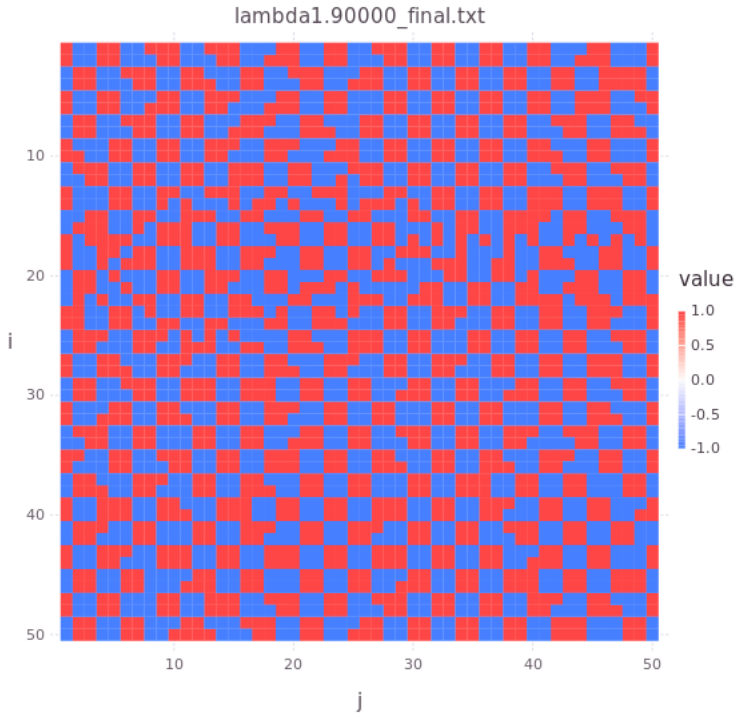
(a) $\lambda = 1.76139$ con quiralidad



Azul claro: ⌞ (izquierda)
Azul oscuro: ⌝ (izquierda)
Rojo claro: ⌟ (derecha)
Rojo oscuro: ⌜ (derecha)

(b) $\lambda = 1.76139$ con quiralidad y orientación

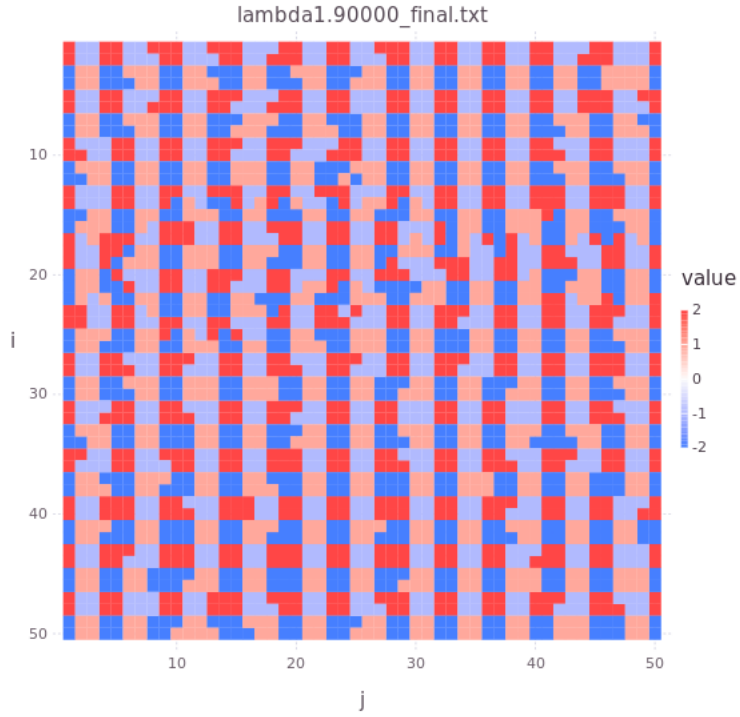
Figura 6.15: Sistema con $\lambda = 1.76139$.



Azul :L o ⅂ (izquierdas)

Rojo: ⅃ o ⅂ (derechas)

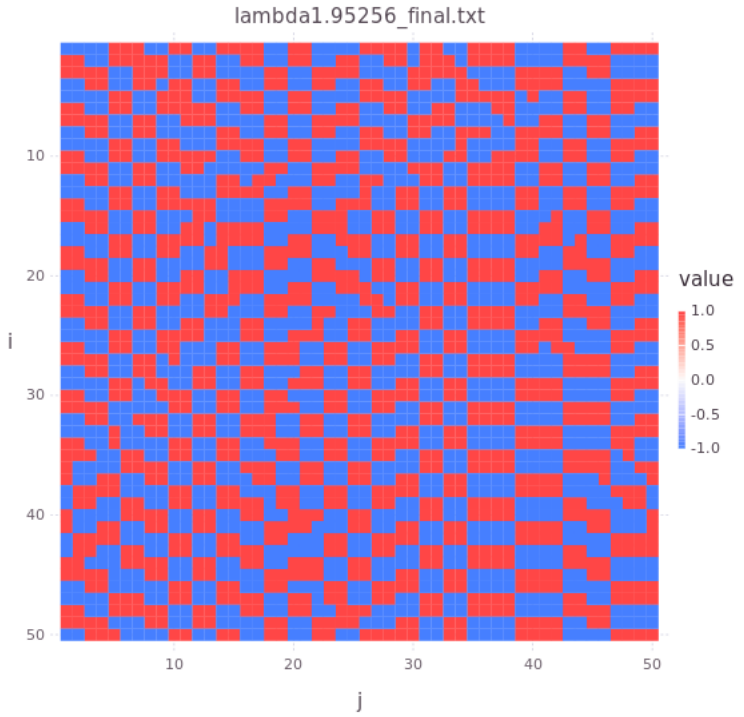
(a) $\lambda = 1.9$ con quiralidad



Azul claro: L (izquierda)
Azul oscuro: ⅂ (izquierda)
Rojo claro: ⅃ (derecha)
Rojo oscuro: ⅂ (derecha)

(b) $\lambda = 1.9$ con quiralidad y orientación

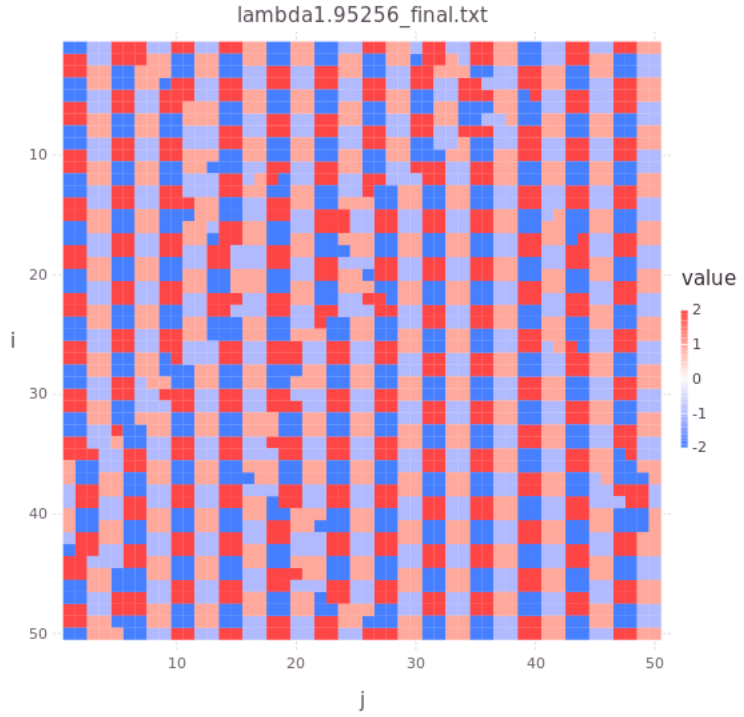
Figura 6.16: Sistema con $\lambda = 1.9$.



Azul :L o ⅂ (izquierdas)

Rojo: ⅃ o ⌜ (derechas)

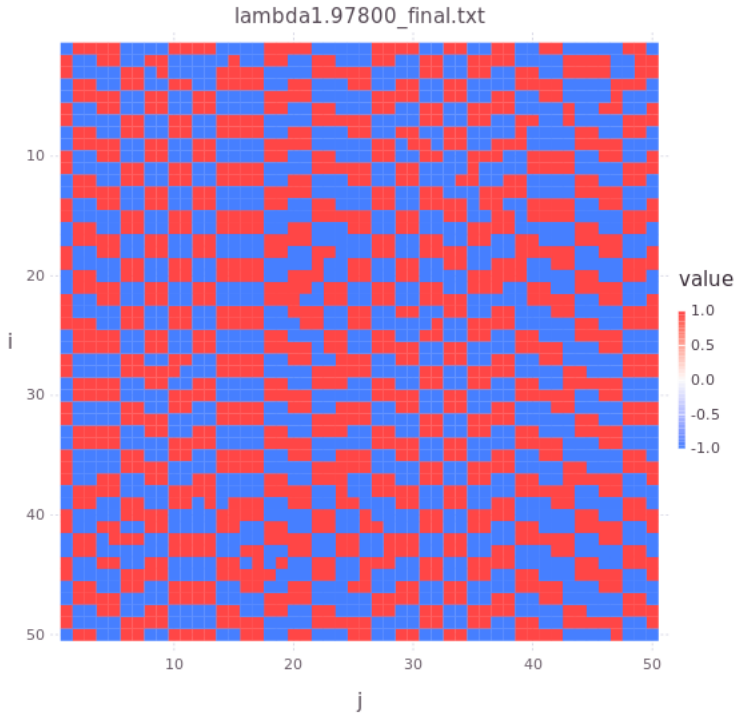
(a) $\lambda = 1.95256$ con quiralidad



Azul claro: L (izquierda)
Azul oscuro: ⅂ (izquierda)
Rojo claro: ⅃ (derecha)
Rojo oscuro: ⌜ (derecha)

(b) $\lambda = 1.95256$ con quiralidad y orientación

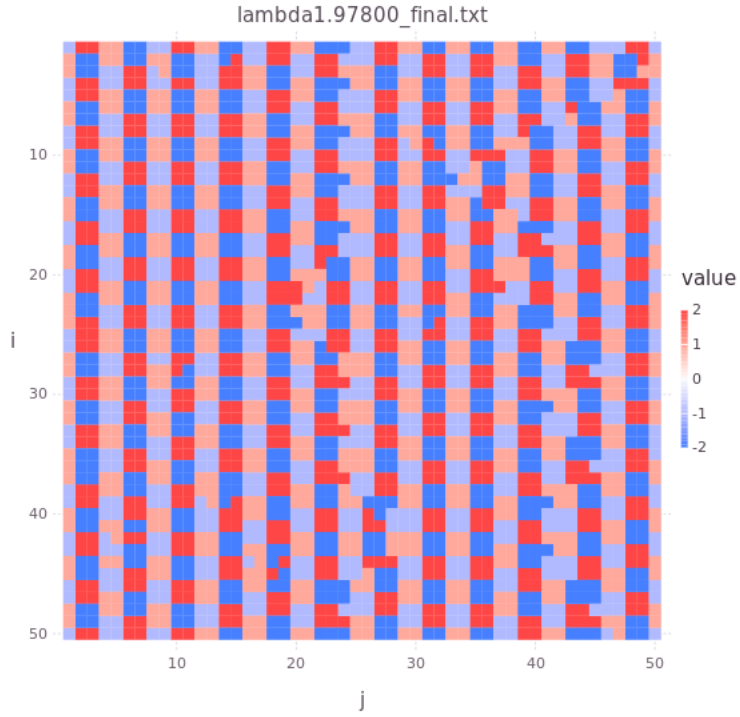
Figura 6.17: Sistema con $\lambda = 1.95256$.



Azul :L o ⅂ (izquierdas)

Rojo: ⅃ o ⅂ (derechas)

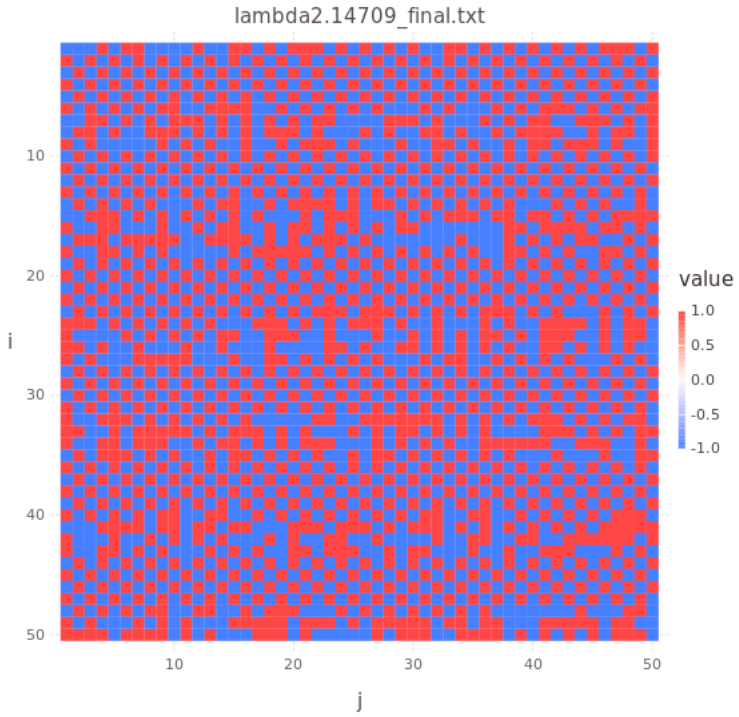
(a) $\lambda = 1.97800$ con quiralidad



Azul claro: L (izquierda)
Azul oscuro: ⅂ (izquierda)
Rojo claro: ⅃ (derecha)
Rojo oscuro: ⅂ (derecha)

(b) $\lambda = 1.97800$ con quiralidad y orientación

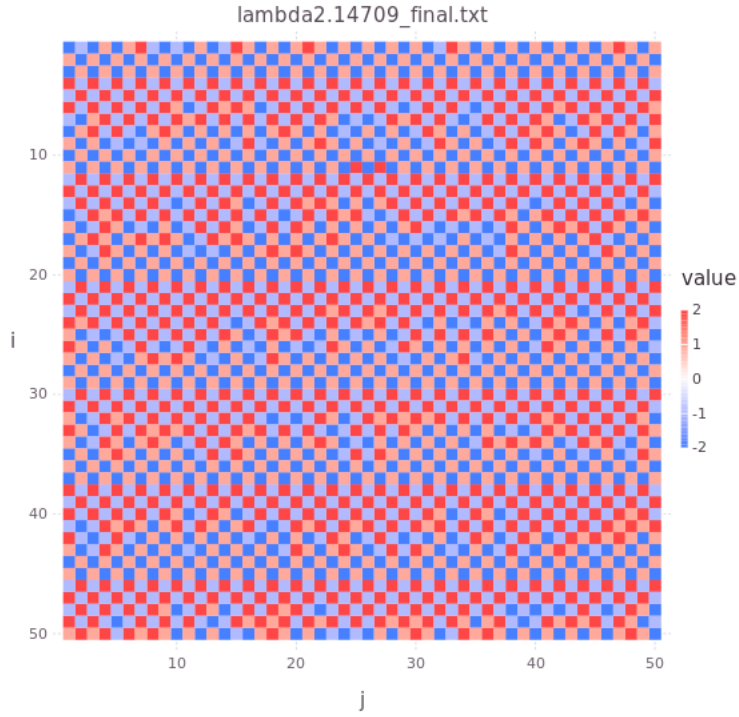
Figura 6.18: Sistema con $\lambda = 1.97800$.



Azul :L o ⅂ (izquierdas)

Rojo: ⅃ o ⌜ (derechas)

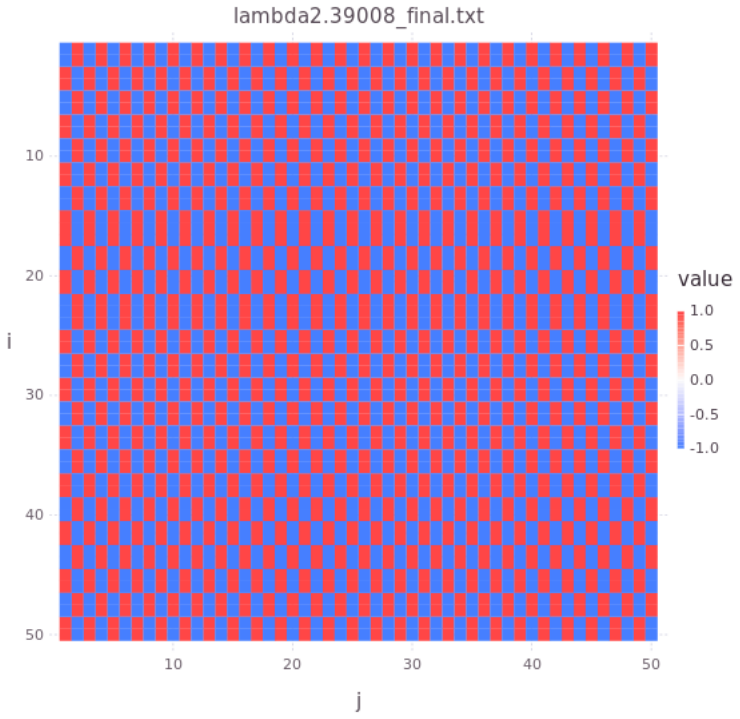
(a) $\lambda = 2.14709$ con quiralidad



Azul claro: L (izquierda)
Azul oscuro: ⅂ (izquierda)
Rojo claro: ⅃ (derecha)
Rojo oscuro: ⌜ (derecha)

(b) $\lambda = 2.14709$ con quiralidad y orientación

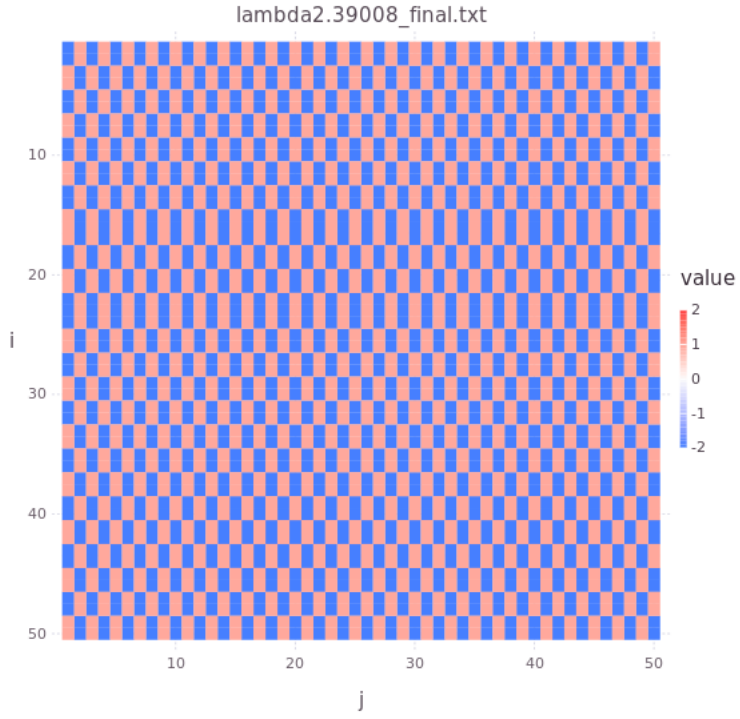
Figura 6.19: Sistema con $\lambda = 2.14709$.



Azul :L o ⅂ (izquierdas)

Rojo: ⅃ o ⌜ (derechas)

(a) $\lambda = 2.39008$ con quiralidad



Azul claro: L (izquierda)
Azul oscuro: ⅂ (izquierda)
Rojo claro: ⅃ (derecha)
Rojo oscuro: ⌜ (derecha)

(b) $\lambda = 2.39008$ con quiralidad y orientación

Figura 6.20: Sistema con $\lambda = 2.39008$.

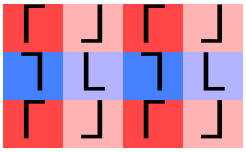


Tabla 6.1: Vecinos de L para $\lambda = 0.55$.

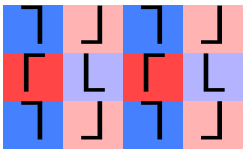


Tabla 6.2: Vecinos de L para $\lambda = 0.55902$.

Como se ha podido apreciar en las figuras anteriores, la mayoría posee algún patrón en la distribución espacial de las partículas, que es más evidente en las figuras que muestran quiralidad y orientación. Para los primeros valores de λ (es decir, para $\lambda = 0.1, 0.46098$) parece haber dos patrones: barras verticales del mismo tipo de partícula (de la misma quiralidad y orientación) y partículas alternándose con otras de otro tipo.

Para $\lambda = 0.55$ el patrón parece ser el siguiente: dada una partícula, arriba y abajo tiene partículas de la misma orientación pero de quiralidad opuesta; a los lados las partículas son de la misma quiralidad pero de orientación opuesta y en las diagonales son de quiralidad y orientación opuestas. Por ejemplo, si la partícula es de tipo L, la distribución de sus vecinos es la que se muestra en la Tabla 6.1.

Para $\lambda = 0.55902$ el patrón predominante es algo similar al de $\lambda = 0.55$. Dada una partícula, arriba y abajo tiene partículas de la misma orientación pero de quiralidad opuesta; a los lados las partículas son de quiralidad y orientación opuestas y en las diagonales son de la misma quiralidad pero de orientación opuesta. Por ejemplo, si la partícula es de tipo L, la distribución de sus vecinos es la que se muestra en la Tabla 6.2. La similitud con $\lambda = 0.55$ proviene de que si, para una partícula, se recorren las bandas verticales que están a los lados un lugar arriba o abajo, se obtiene el patrón de $\lambda = 0.55902$.

Cuando $\lambda = 1.0$ se generan estados enantiopuros o se generan bandas verticales (alineadas con el lado largo de las partículas) de ancho variable de partículas en el que se alternan partículas de quiralidad opuesta. A su vez, en estas regiones hay otras bandas de ancho variable de partículas con la misma quiralidad pero de orientación opuesta.

Para los valores de $\lambda = 1.00499, 1.05475, 1.09659$ se obtienen regiones con bandas verticales de una partícula de ancho, donde cada banda está compuesta de un solo tipo de partículas y las dos bandas vecinas de partículas con quiralidad y orientación opuestas. Un ejemplo se muestra en la Tabla 6.3

En los sistemas con $\lambda = 1.14127, 1.34536$ también se generan bandas verticales de una partícula de ancho, pero el patrón de alternancia entre bandas no es tan claro como en el ejemplo anterior. Además, para $\lambda = 1.34536$, parece haber líneas diagonales que dividen el sistema.

Los sistemas con $\lambda = \sqrt{2} \approx 1.41421$ generan estados enantiopuros, en los cuales las partículas se organizan en bandas diagonales de partículas con la misma orientación. Es importante observar que las bandas de un estado enantiopuro son ortogonales a la de un estado con la quiralidad opuesta. Las

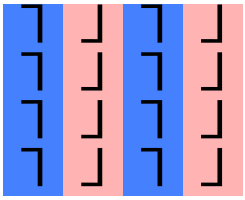


Tabla 6.3: Bandas verticales con partículas de distinta quiralidad y orientación

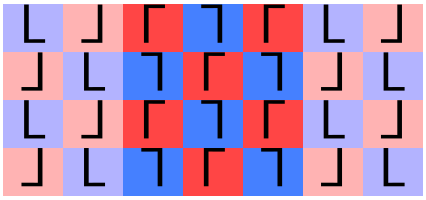


Tabla 6.4: Bandas con partículas alternadas de orientación similar y quiralidad opuesta.

bandas de las partículas izquierdas, coloreadas con azul, van de arriba-izquierda a abajo-derecha, como $\diagdown$. Las de las partículas derechas (en rojo) van de arriba-derecha a abajo-izquierda, como $\diagup$.

Para $\lambda = 1.45, 1.45344$ se obtienen dos configuraciones mezcladas. Por una parte, hay regiones como las de $\lambda = \sqrt{2}$, aunque son pequeñas. Por otra parte, hay bandas en las que en su interior se alternan partículas de quiralidad opuesta y orientación similar. Estas últimas se alternan con bandas donde las partículas poseen la orientación contraria. Las bandas se orientan de manera vertical. Un ejemplo se muestra en la Tabla 6.4.

Para $\lambda = 1.70660$ se forman bandas más o menos horizontales de partículas que alternan quiralidad y orientación. El patrón es más claro para $\lambda = 2.14709$.

En los sistemas donde $\lambda = 1.9, 1.95256, 1.97800$ y en menor medida para $\lambda = 1.76169$ ocurre la formación de algo parecido a un tablero de ajedrez. Se parece a la Tabla 6.1 pero cada celda del "tablero" está compuesto por cuatro partículas del mismo tipo. La Tabla 6.5 describe mejor la situación.

En los sistemas donde $\lambda = 2.14709$ se forman bandas más o menos horizontales y de ancho variable. Cada banda se compone de partículas que se alternan en quiralidad y orientación. El

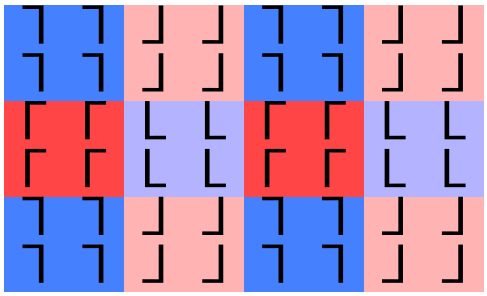


Tabla 6.5: Configuración para sistemas con$\lambda = 1.9, 1.95256, 1.97800$.

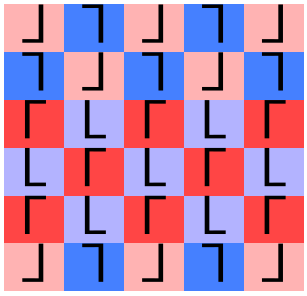


Tabla 6.6: Bandas horizontales del sistema con $\lambda = 2.14709$.

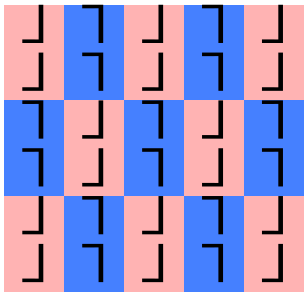
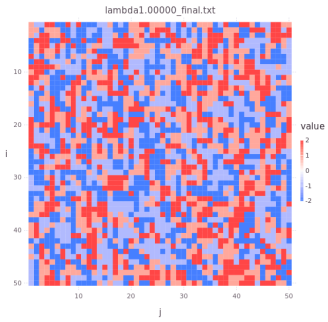


Tabla 6.7: Configuración de tablero alargado para $\lambda = 2.39008$.

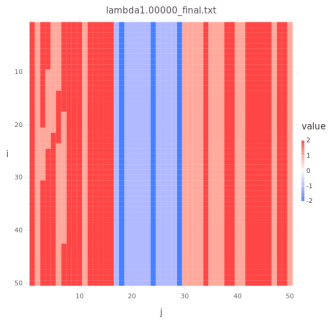
cuadro 6.6 muestra dichas bandas.

Finalmente, para los sistemas donde $\lambda = 2.39008$ la configuración parece como un tablero de ajedrez pero con celdas alargadas verticalmente. Cada celda es de una partícula de ancho, pero es de dos de largo. Las celdas del tablero se componen de partículas de quiralidad y orientación opuesta. La Tabla 6.7 muestra dicha configuración.

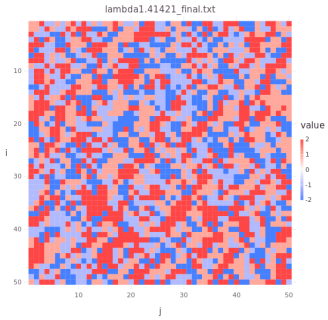
Como se mencionó, a partir de $T = 0.4$ los efectos térmicos comienzan a ser importantes para algunos sistemas. Las configuraciones son menos definidas que a las temperaturas bajas, aunque eso se nota más para valores de $\lambda < 1.7$. Como se puede observar en la Fig. 6.21, los patrones son menos claros.



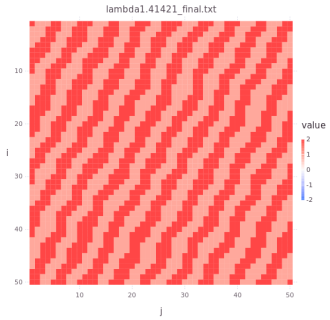
(a) Sistema con $\lambda = 1.0$ y $T = 1.0$.



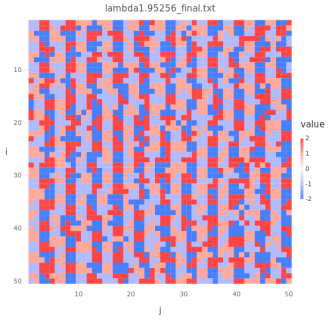
(b) Sistema con $\lambda = 1.0$ y $T = 0.1$.



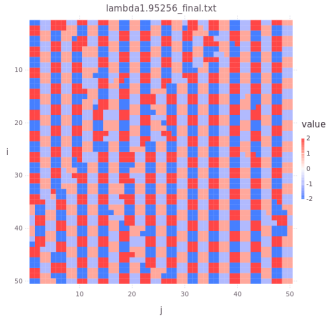
(c) Sistema con $\lambda = 1.41421$ y $T = 1.0$.



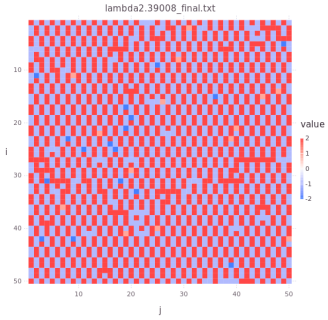
(d) Sistema con $\lambda = 1.41421$ y $T = 0.1$.



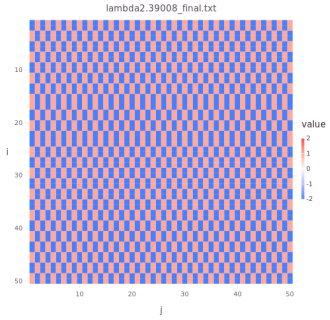
(e) Sistema con $\lambda = 1.95256$ y $T = 1.0$.



(f) Sistema con $\lambda = 1.95256$ y $T = 0.1$.



(g) Sistema con $\lambda = 2.39008$ y $T = 1.0$.



(h) Sistema con $\lambda = 2.39008$ y $T = 0.1$.

Figura 6.21: Comparación de sistemas a $T = 1.0$ y $T = 0.1$.

Capítulo 7

Análisis

Para averiguar si el modelo es capaz de generar separación quiral de algún tipo se hicieron dos análisis. El primero fue comparar las energías de los sistemas obtenidas en las simulaciones con los de algunos estados. El segundo fue identificar si ocurría algún tipo de separación quiral.

7.1. Energías

En el primer estudio, para cada valor de λ se comparan las energías obtenidas de las simulaciones a $T = 0.1$ con las energías de las siguientes configuraciones. Se eligió $T = 0.1$ porque a esa temperatura se forman la mayoría de patrones de interés.

- Aleatoria: Las posiciones de las partículas se asignan de manera aleatoria, como las configuraciones iniciales de las simulaciones en el ensamble canónico (es decir, con el sistema totalmente ocupado).
- Enantiopura: todas las partículas poseen la misma quiralidad. En este caso se consideran los siguientes tipos:
  - Misma orientación (abreviada MO)
  - Alternada entre diferente orientación (abreviada DO)
  - Bandas horizontales de diferente orientación (abreviada BH)
  - Bandas verticales de diferente orientación (abreviada BV)

La energía de la configuración de la simulación se compara la de una configuración aleatoria para estar seguros de que los patrones obtenidos no hayan sido aleatorios. En el ensamble canónico la probabilidad de encontrar un estado i es proporcional a $e^{-\beta E_i}$. En el gran canónico es proporcional a $e^{-\beta N_i(E_i-\mu)}$. Con el mismo número de partículas, si una configuración tiene la misma energía que cualquier configuración aleatoria, las dos tienen la misma probabilidad de ocurrir, por lo que también es aleatoria.

También se compara con distintas configuraciones enantiopuras para saber si alguna simulación ha quedado arrestada en una configuración de mayor energía a la enantiopura. Si la energía de la simulación es mayor, existe la posibilidad de que sus estados finales sean enantiopuros. Si es

λ	Simulación	Aleatorio	MO	Alternados	BH	BV
0.1	-1.4976	-1.2584	-1.0	-1.0	-1.0	-1.0
0.46098	-2.4992	-2.3544	-2.0	-2.5	-2.0	-2.5
0.55	-3.9988	-3.5636	-3.0	-3.5	-3.0	-3.5
0.55902	-5.4908	-4.7672	-4.0	-4.5	-4.5	-5.0
1.0	-9.9976	-9.3356	-10.0	-8.5	-9.5	-10.0
1.00499	-12.5	-11.6148	-12.0	-10.5	-11.5	-12.0
1.05475	-13.5	-12.7032	-13.0	-12.0	-13.0	-13.0
1.09659	-15.5	-14.88	-15.0	-14.0	-15.0	-15.0
1.14127	-19.0	-18.3124	-18.0	-17.0	-18.0	-18.0
1.34536	-23.72	-22.82	-22.0	-21.5	-22.5	-22.0
1.41421	-28.1	-27.3424	-28.0	-27.5	-26.5	-26.0
1.45	-29.2	-28.4756	-29.0	-28.5	-27.5	-27.0
1.45344	-30.2024	-29.5596	-30.0	-29.5	-29.0	-28.5
1.7066	-36.0927	-35.2836	-35.0	-35.5	-34.0	-34.0
1.76139	-38.208	-37.5060	-37.0	-37.5	-36.5	-36.0
1.9	-44.1184	-43.1568	-42.0	-43.0	-42.0	-41.0
1.95256	-46.6444	-45.4303	-44.0	-45.5	-44.0	-43.5
1.978	-47.7724	-46.5624	-45.0	-46.5	-45.5	-45.0
2.14709	-61.1196	-60.0572	-59.0	-60.5	-59.5	-59.0
2.39008	-76.44	-74.7316	-76.0	-73.5	-75.0	-74.5

Tabla 7.1: Energía por partícula de diversas configuraciones con distintos rangos de interacción

menor, podría ser útil compararla con alguna configuración enantiopura más compleja que las que se utilizaron aquí, aunque el algoritmo de Metropolis debería ser capaz de encontrar las configuraciones de menor energía.

La distinción entre bandas horizontales y verticales es importante debido a la forma de las partículas. Las bandas verticales son las que están alineadas a los lados largos de las partículas. Las bandas horizontales están alineadas a los lados cortos. Por estas razones es posible distinguir dos orientaciones distintas en el sistema.

Las energías (por partícula) de estas configuraciones se muestran en la Tabla 7.1. La primera columna muestra el valor de λ que se tomó para cada conjunto de configuraciones. Las celdas resaltadas muestran cuál es la configuración que posee la menor energía. La Tabla 7.1 muestra los resultados de este análisis.

Como se puede ver en la Tabla 7.1, casi todas las simulaciones generaron configuraciones con menores energías que las configuraciones probadas. Los únicos dos casos son cuando $\lambda = 0.46098, 1.0$, por lo que es posible que los estados de mínima energía son enantiopuros.

7.2. Separación quiral

El segundo análisis consiste en averiguar si hay separación quiral. En este caso se utilizaron dos criterios para determinar si hay algún tipo de separación. El primero fue utilizando el exceso enantiomérico. El segundo fue contar el número de cúmulos de partículas de cierto tamaño.

Figura 7.1: Gráfica de exceso enantiomérico respecto a la temperatura.

7.2.1. Exceso enantiomérico

El exceso enantiomérico sirve para saber si la mayoría de las moléculas de una mezcla racémica poseen al misma quiralidad. Retomando la definición que se dio en la introducción, el exceso enantiomérico es

$$e.e = \frac{|N_I - N_D|}{N_I + N_D}.$$

Si $e.e = 0$ hay exactamente el mismo número de partículas derechas e izquierdas. Si $e.e = 1$ hay un solo tipo de enantiómero. En este caso, se consideró que hay separación quiral si $e.e >= 0.85$.

La Fig. 7.1 muestra el exceso enantiomérico con respecto a la temperatura. La línea roja marca $e.e. = 0.85$. Los puntos marcan los resultados de la simulación. Las líneas que los unen son solo una guía visual. En la Fig. se puede observar que solamente para $\lambda = 1.0$ (color lila) y $\sqrt{2} \approx 1.41421$ (color verde olivo) existen sistemas con $e.e. >= 0.85$ a temperaturas bajas. La otra línea que sobresale un poco en la Fig. es la de $\lambda = 1.45344$. Para $T <= 0.5$, en esos sistemas existen pequeñas regiones enantiopuras que explican su $e.e.$ más alto que el de la mayoría.

También se comparó el $e.e.$ entre sistemas de distinto tamaño. Los tamaños fueron de 50×50, 100×100 y 200×200 a $T = 0.1$, aunque las simulaciones fueron más cortas que las anteriores, ya que para los sistemas más grandes las simulaciones tardaban demasiado. La Fig. 7.2 muestra que, en general, el exceso enantiomérico baja conforme sube el tamaño del sistema. Sin embargo, este resultado puede deberse a que como las simulaciones fueron más cortas, puede haber sistemas que aún no alcanzan el equilibrio térmico. Esto sería más relevante conforme aumente el tamaño del

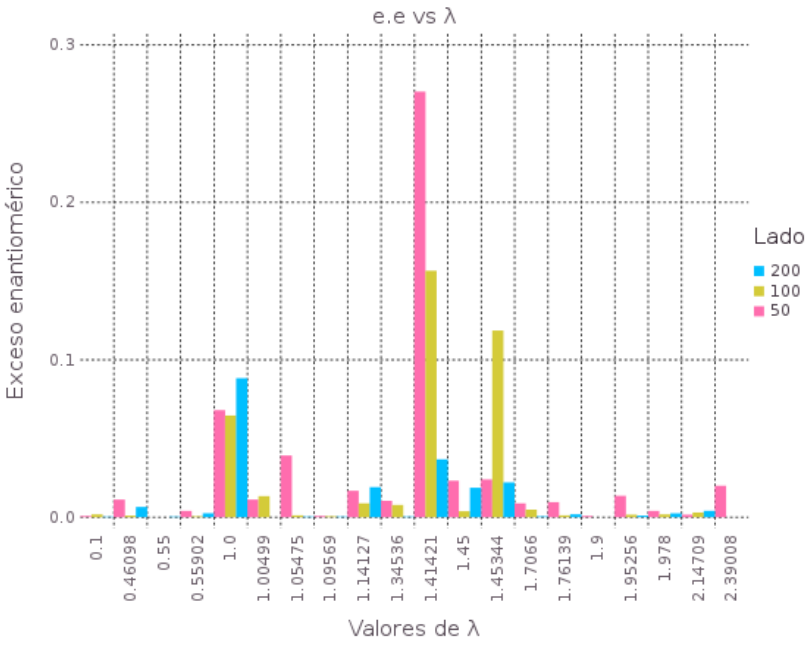


Figura 7.2: Gráfica de exceso enantiomérico respecto a λ, dependiendo del tamaño del sistema.

sistema, porque le faltaría mucho más para alcanzar el equilibrio.

7.2.2. Número y tamaño de cúmulos

El segundo criterio consistió en saber cuántos cúmulos de enantiómeros hay y de qué tamaño son. La idea es que el exceso enantiomérico puede ser bajo pero los enantiómeros se encuentren agrupados en una parte del sistema. Si todas las moléculas son del mismo enantiómero, hay un solo cúmulo. Si hay pocas moléculas del otro enantiómero, habrá un cúmulo grande y algunos pequeños. Puede que haya pocos cúmulos grandes (por decir, cinco o menos) que abarquen la mayor parte del sistema.

Cuando el exceso enantiomérico sea menor a 0.85 pero haya al menos un cúmulo de partículas con la misma quiralidad que abarque al menos el 10 % de las partículas de todo el sistema se dirá que hay **microsegregación quiral**.

En la Fig. 7.3 se muestra la fracción que ocupa el cúmulo más grande de un sistema. Los veinte valores de λ se muestran en cuatro gráficas separadas para mayor claridad y la escala en el eje y es semilogarítmica. La línea horizontal naranja marca el 5 % del sistema y la roja el 10 %, así que los sistemas por encima de la línea roja presentan microsegregación a la temperatura indicada.

En la Fig. 7.3 se puede observar que los sistemas que presentan microsegregación en casi todo el rango de temperaturas son $\lambda = 1.0, 1.41421$. Esto es comprensible dado que son los que también tienen segregación quiral. Aunque no tengan el criterio de $e.e. >= 0.85$, aún hay muchas partículas de la misma quiralidad.

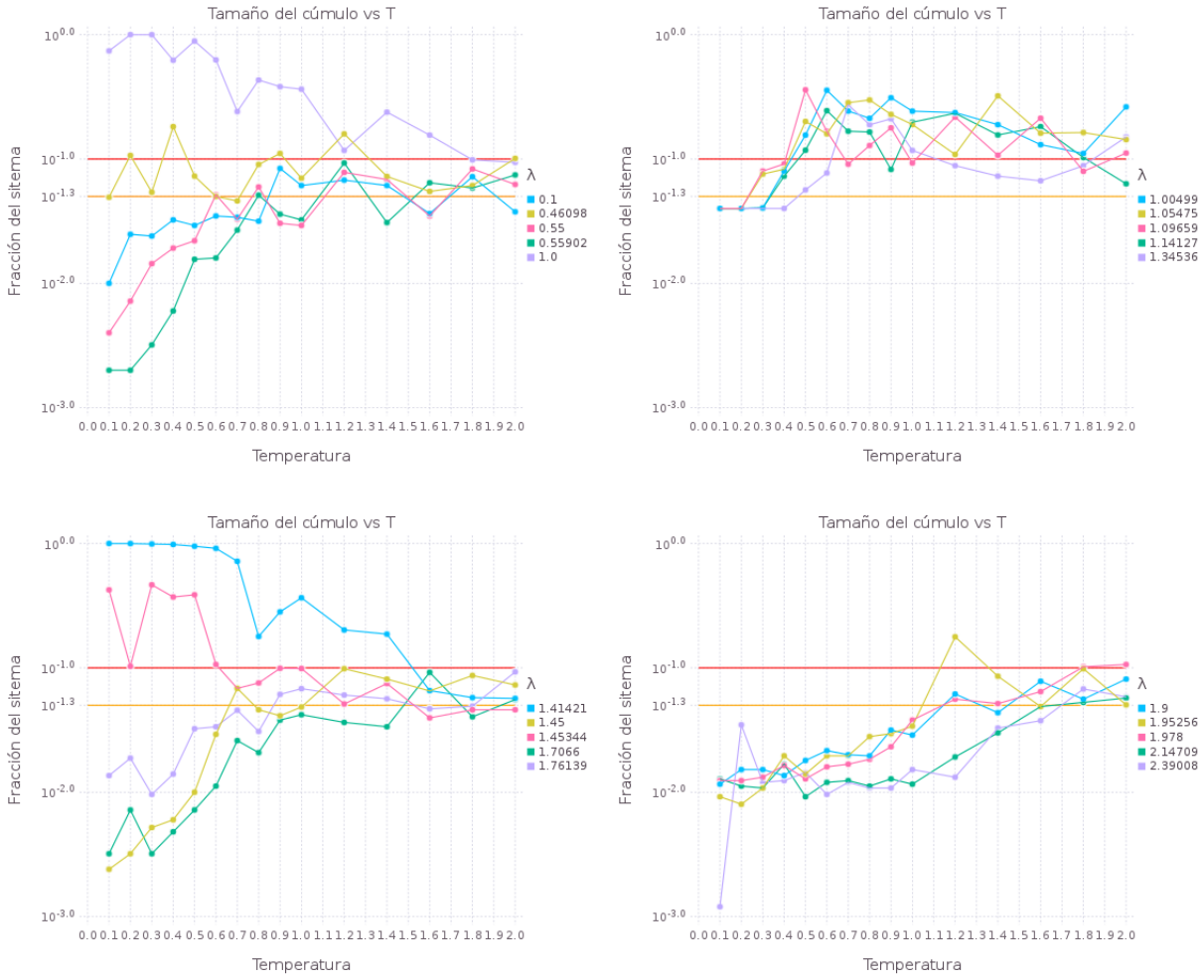
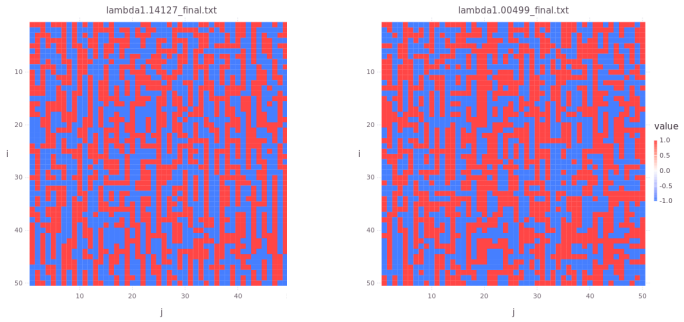


Figura 7.3: Gráficas en escala semilogarítmica del tamaño del cúmulo más grande de un sistema con respecto a la temperatura.



(a) Sistema con $\lambda = 1.14127$. (b) Sistema con $\lambda = 1.00499$.

Figura 7.4: Sistemas a T = 0.8. Se muestra únicamente la quiralidad.

El sistema con $\lambda = 1.45344$, que también posee un *e.e.* ligeramente alto, igualmente presenta microsegregación en el rango $T = [0.1, 0.6]$

Las novedades son los sistemas con $\lambda = 1.00499, 1.05475, 1.09659, 1.14127$, que presentan microsegregación a partir de $T = 0.5$. La Fig. 7.4 muestra dos sistemas a $T = 0.8$, donde se puede observar la formación de cúmulos de la misma quiralidad.

El resto de los sistemas solo presentan picos de microsegregación o no poseen a ninguna temperatura.

Capítulo 8

Conclusiones

Se concluye que, utilizando el criterio de exceso enantiomérico, los sistemas con $\lambda = 1.0, 1.41421$ presentan separación quiral a bajas temperaturas. Utilizando el criterio de microsegregación, los sistemas anteriores presentan microsegregación en casi todo el rango de temperaturas, además de que $\lambda = 1.00499, 1.05475, 1.09659, 1.14127, 1.45344$ la presentan presentan a partir de T = 0.5.

El comportamiento del modelo resulto mucho más complicado de lo que se pensó en un principio. Al principio se esperaba que fuera un poco parecido al modelo de Ising, pero el comportamiento del modelo propuesto en este trabajo es muy distinto. Además varía notablemente dependiendo del valor de λ asignado.

Los patrones de partículas descritos después de las figuras de los sistemas fueron los más sencillos de identificar visualmente. Sin embargo, puede que haya patrones más complicados de descubrir y que requieran de técnicas más sofisticadas.

Sería conveniente realizar las simulaciones completas con sistemas de varios tamaños para observar si ocurren cambios. El problema es que, para sistemas de al menos $100\times$ 100, las simulaciones comienzan a tardar mucho. Quizá paralelizar la simulación de un solo sistema disminuiría el tiempo.

Hay varias modificaciones que se podrían hacer a este modelo. Por ejemplo, se fijó $l_1 = 0.9$ y $l_2 = l_1/2$. Se podrían cambiar las proporciones entre ambos segmentos. Incluso con la misma proporción, se podría investigar si los patrones cambian al utilizar longitudes distintas. Otra modificación posible es permitir que las partículas roten de manera más libre. Una variante es que el lado largo pueda alinearse horizontalmente, por lo que una partícula de tipo ⌞ puede tener los estados ⌞, ⌙, ⌝, ⌜. La otra variante interesante es dejarla rotar libremente alrededor de su sitio de red. La más ambiciosa es dejar que las partículas se muevan libremente. Se podría observar qué efectos comparte el modelo con [16] y qué fenómenos nuevos surgen.

Apéndice A

Equilibración

Es muy común que cuando se realizan simulaciones de Montecarlo se les deje correr un tiempo antes de comenzar a calcular las propiedades macroscópicas del sistema. Esto es debido a que el estado del sistema con el que se inicia la simulación no necesariamente es representativo de los estados con las propiedades termodinámicas que se quieren simular. Por ejemplo, en una simulación del modelo de Ising en el ensamble canónico, cuando se inicia con un estado a $T = 0$ todos los espines poseen una sola orientación y a $T = \infty$ todos poseen orientaciones aleatorias; pero lo que interesa estudiar es el sistema a una $T > 0$ finita, donde los estados más probables no son necesariamente los iniciales. Cuando se alcanzan dichos estados probables a una temperatura dada, se dice que el sistema ha alcanzado el **equilibrio**.

En general, se dice que un sistema (en el ensamble canónico) está en equilibrio cuando la probabilidad de encontrar a un sistema en algún estado μ es proporcional al peso de Boltzmann del estado, $p_\mu = e^{-\beta E_\mu}$. Los estados más probables son los que poseen una p_μ alta, que la mayoría de las veces serán aceptados por el algoritmo de Metropolis y que finalmente determinan las propiedades macroscópicas del sistema. El rango de valores que efectivamente adquieren estas propiedades macroscópicas es muy pequeño comparado con el rango más grande posible.

Una manera gráfica de observar que un sistema ha alcanzado el equilibrio es calcular algunas de sus propiedades a lo largo de la simulación y graficarlas contra el tiempo que le toma alcanzarlo. Es práctica usual calcular dicho tiempo como el número de pasos de Montecarlo dividido por el número de partículas N, para que así sea independiente del tamaño del sistema. Se dice que cuando la simulación completa N pasos de Montecarlo, se ha realizado un **barrido** del sistema. En cada barrido del sistema se calculan las propiedades necesarias para observar el equilibrio.

Al graficar las propiedades a lo largo del tiempo, se puede observar que el sistema alcanza el equilibrio cuando sus propiedades fluctúan relativamente poco alrededor de un solo valor constante. Al periodo que le toma alcanzar el equilibrio se le conoce como **tiempo de equilibración** τ_{eq}. Las fluctuaciones, por supuesto, provienen de las pequeñas variaciones permitidas por la temperatura entre las energías de los distintos estados del sistema. Por ejemplo, en una simulación del modelo de Ising a $T < T_c$ se puede observar que, después de varios barridos, los valores de la magnetización fluctúan poco alrededor de un solo valor. Se asume que ese es el valor de la magnetización en el equilibrio. Lo mismo ocurre con la energía por espín.

Como no siempre se conoce el valor de una propiedad en equilibrio, es común realizar varias simulaciones con las mismas condiciones (por ejemplo, temperatura) pero con distintas condiciones

iniciales. Esto se hace porque el sistema podría terminar en un mínimo local de energía, donde también sus propiedades macroscópicas varían muy poco, pero no necesariamente corresponden a las propiedades del mínimo global de energía. Repetir las simulaciones dos o tres veces con condiciones iniciales distintas (pero manteniendo fijos los parámetros de interés) es una manera práctica (pero no infalible) de hallar los estados de mínima energía global y con ello, las propiedades del sistema en equilibrio.

Apéndice B

Unidades reducidas

En algunas simulaciones puede ser muy útil el uso de unidades reducidas. Esto significa que una o varias propiedades se miden con respecto a algunos parámetros intrínsecos del sistema [11]. Por ejemplo, el potencial de Lennard-Jones

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

tiene dos parámetros: σ, la distancia a la que el potencial se vuelve cero; y ϵ, la profundidad energética del pozo del potencial. Si se utilizan σ y ϵ como unidades de medida, el potencial se simplifica a

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

donde $u^* = u/\epsilon$ es el potencial reducido y $r^* = r/\sigma$ es la distancia reducida. De esta manera, las energías (no reducidas) se miden en múltiplos de ϵ y las distancias en múltiplos de σ. Cualquier otra cantidad relacionada con energías o distancias también quedará en términos de σ y ϵ; en particular la temperatura reducida es $T^* = kT/\epsilon$. Despejando T se ve que se puede medir en unidades de ϵ/k. Para evaluar en unidades del SI u otro sistema, se fijan σ y ϵ a los valores deseados.

La posibilidad de usar unidades reducidas cuando se utiliza un potencial con la forma $u(r) = \epsilon f(\sigma/r)$, como el potencial de Lennard-Jones o el de pozo cuadrado (usado en las simulaciones de este trabajo), está sustentado en que los valores de los parámetros σ y ϵ se pueden agrupar en familias. Los valores contenidos en una familia generan exactamente el mismo estado reducido. Los resultados de la simulación de un estado reducido particular sirven para la familia de parámetros que representa, con lo cual es posible evitar duplicar la misma simulación [11, 12].

En algunos casos hay un cierto ahorro de tiempo de cómputo porque se eliminan algunas multiplicaciones o divisiones. Por ejemplo, al implementar el potencial reducido u^* se evita el cálculo de σ/r para cada r y simplemente se trabaja desde el inicio con r^*. También se evita la multiplicación de ϵ. Al final los datos más relevantes (en vez de todos los datos generados) de la simulación son los que se convierten a unidades no reducidas. La ventaja es menos clara si hay múltiples propiedades y la mayoría no se puede usar como unidad reducida básica [12].

Una consecuencia de utilizar unidades reducidas es que la magnitud de varios de los parámetros medidos durante una simulación es de orden de la unidad; dichas magnitudes se encuentran aproximadamente en el rango $[10^{-3}, 10^{3}]$, lo que presenta dos ventajas. La primera es que un número

muy alejado de ese rango puede indicar un error. La segunda es que disminuye la posibilidad de que se introduzcan errores de tipo *overflow* o *underflow* [11].

Apéndice C

Representación de números en la computadora

En este apéndice se hablará brevemente de la manera en la que las computadoras representan los números, particularmente a los números reales. Lo más importante es que se representan en base binaria, con unos y ceros; que se utiliza la notación de punto flotante porque permite representar grandes cantidades de números y que la representación no es perfecta pero es útil.

C.1. Sistema de numeración binario

C.1.1. Sistemas de numeración posicionales

Los números se representan con símbolos especiales llamados **dígitos**. Actualmente y en la mayor parte del mundo, los dígitos se organizan utilizando un **sistema de numeración posicional**, definido por una **base** o **raíz** b y en el cual cada dígito adquiere un valor relativo dependiendo de su posición respecto de otro símbolo llamado **punto radical**, ubicado entre a_0 y a_{-1}. Los valores debido a las posiciones se definen por [18]

$$(\cdots a_3 a_2 a_1 a_0 . a_{-1} a_{-2} \ldots)_b = \cdots + a_3 b^3 + a_2 b^2 + a_1 b^1 + a_0 + a_{-1} b^{-1} + a_{-2} b^{-2} + \cdots$$

En el sistema decimal, que es el más utilizado por los seres humanos, $b = 10$, los dígitos son 0, 1, 2, 3, 4, 5, 6, 7, 8 y 9 y al punto radical se le llama punto decimal. Las computadoras utilizan otros sistemas. El más utilizado es el binario, donde $b = 2$ y solo utiliza los dígitos 0 y 1. Se utiliza el sistema binario porque se puede implementar directamente con compuertas lógicas. Otro sistema de numeración utilizado es el hexadecimal, donde $b = 16$ e incluye los dígitos del 0 al 9 y A, B, C, D, E, F. Otro más es el octal, con $b = 8$ y que utiliza los dígitos del 0 al 7.

C.1.2. Conversión de base binaria a decimal

Es posible convertir números de una base a otra. Es necesario que la computadora pueda entender y proporcionar números en base decimal, ya que trabaja en base binaria.

Para convertir números en base 2 a base 10, simplemente se utiliza la regla dada anteriormente para definir los valores relativos de los dígitos [19]. Por ejemplo, para convertir el número $(101110.001)_2$ a base decimal:

$$(101110.001)_2 = 1*2^5 + 0*2^4 + 1*2^3 + 1*2^2 + 1*2^1 + 0 + 0*2^{-1} + 0*2^{-2} + 1*2^{-3} =$$

$$32 + 8 + 4 + 2 + 0.125 = (46.125)_{10} = 46.125$$

También se puede usar el siguiente algoritmo recursivo para convertir números enteros en base b a base 10:

$$\begin{aligned} d_n &= a_n \\ d_{n-1} &= a_{n-1} + d_n b \\ d_{n-2} &= a_{n-2} + d_{n-1} b \\ &\cdots\cdots\cdots\cdots \\ d_0 &= a_0 + d_1 b \end{aligned}$$

Entonces d_0 es la representación en base 10. Por ejemplo, convertir $(101110)_2$:

$$\begin{aligned} d_n &= 1 \\ d_{n-1} &= 0 + 1*2 = 2 \\ d_{n-2} &= 1 + 2*2 = 5 \\ d_{n-3} &= 1 + 5*2 = 11 \\ d_{n-4} &= 1 + 11*2 = 23 \\ d_{n-5} &= 0 + 23*2 = 46 \end{aligned}$$

C.1.3. Conversión de base decimal a binaria

Para convertir números en base 10 a base 2 se puede seguir el siguiente algoritmo. Para la parte entera, se divide el número entero entre la base y el residuo de la división es el primer dígito a la derecha en base 2. Se repite el algoritmo hasta que el resultado de la división sea 1 o 0. Cada residuo corresponde al siguiente dígito del número en base 2. Finalmente, el último cociente corresponde al dígito más significativo del número entero. Por ejemplo, la representación binaria de $46 = (46)_{10}$ es:

$$\begin{aligned} 46/2 &= 23 \text{ y sobra } 0 \\ 23/2 &= 11 \text{ y sobra } 1 \\ 11/2 &= 5 \text{ y sobra } 1 \\ 5/2 &= 2 \text{ y sobra } 1 \\ 2/2 &= 1 \text{ y sobra } 0 \end{aligned}$$

Y el último cociente es 1. Entonces $46 = (46)_{10} = (101110)_2$.

Para la parte fraccionaria, se multiplica la fracción por 2 y el dígito entero corresponde al primer dígito. Se repite lo mismo con las partes fraccionarias de las multiplicaciones hasta que la

multiplicación dé 1 o se repita la fracción, en cuyo caso se trata de una representación periódica. Por ejemplo, $0.875 = (0.875)_{10} = (0.111)_2$:

$$0.875 * 2 = 1.75$$
$$0.75 * 2 = 1.5$$
$$0.5 * 2 = 1.0$$

Y $0.7 = (0.7)_{10} = (0.\overline{10110})_2$ es periódico porque se repite una parte fraccionaria:

$$0.7 * 2 = 1.4$$
$$0.4 * 2 = 0.8$$
$$0.8 * 2 = 1.6$$
$$0.6 * 2 = 1.2$$
$$0.2 * 2 = 0.4$$

C.2. Notación de punto flotante

Debido a su limitada capacidad de memoria y procesamiento, las computadoras no son capaces de representar todos los números reales de la recta numérica, por lo que es necesario representarlos de alguna manera con un conjunto finito de ellos. Esto se resuelve utilizando la **notación de punto flotante** [20].

Cualquier número real y positivo se puede representar con la notación

$$y = 0.d_1 d_2 \cdots d_p d_{p+1} d_{p+2} \cdots \times b^e$$

donde las d_i representan la parte fraccionaria de y y comúnmente se le conoce como **mantisa**; b representa la base en la que está representado el número (10 para el sistema decimal y 2 para el binario) y e el exponente, también llamado **característica**. A su vez, e tiene la forma $e = e_1 e_2 ... e_l$. Como en este caso la parte entera de y es igual a cero implica que $|y| < 1$; entonces se dice que y está **normalizado** o que y es un **número normal**. Es una de las maneras más comunes de representar a los números flotantes en las computadoras.

C.2.1. Errores de redondeo y aritméticos

Como las computadoras no son capaces de representar todos los dígitos de la mantisa de un número real, es necesario representar de alguna manera los dígitos principales. Una manera es simplemente **truncando** y en los primeros p dígitos. Si $fl(y)$ es la representación de punto flotante de y, entonces $fl(y)$ obtenido con truncado es

$$fl(y) = 0.d_1 d_2 \cdots d_p \times b^e$$

Otra manera es **redondeando** el número. Si $d_{p+1} \geq b/2$ (5 en decimal y 1 en binario), se suma 1 a d_p. Visto de otra manera: al número truncado se le suma $(b/2) \times b^{e-(p+1)}$. Si $d_{p+1} < b/2$, d_p se deja igual (el número solamente ha sido truncado).

A la diferencia que existe entre un número real r y su representación en número flotante r^* se le conoce como **error de redondeo**. Hay dos tipos de error de redondeo:

- Error absoluto: $E = r - r^*$
- Error relativo: $\epsilon = (r - r^*)/|r|$, siempre y cuando $r \neq 0$.

Lo más habitual es utilizar el error relativo para saber si la aproximación r^* es lo suficientemente buena.

Una consecuencia de representar los números en forma de punto flotante es que el error de aproximación relativo posee un límite que es independiente del valor del número a aproximar. Si $r^* = r(1 + \epsilon(r))$ entonces

$$|\epsilon(x)| < b^{1-p} \qquad \text{(truncado)}$$

$$|\epsilon(x)| < \frac{1}{2} b^{1-p} \qquad \text{(redondeo)}$$

Al número b^{1-p} se le denomina **unidad de redondeo** o *ulp* (*unit in the last place*, unidad en el último lugar de la mantisa).

Las operaciones aritméticas también presentan errores, en el sentido de que si x y y son dos números reales y $\circ$ es una operación cualquiera, no siempre se cumple que $x \circ y = fl(x) \circ fl(y)$. Hay dos razones: la primera es que la computadora trabaja con representaciones inexactas de x y y; la segunda es que el resultado de $x^* \circ y^*$ también se tiene que representar con p dígitos, lo que introduce otra fuente de error. En general, el error relativo por cada operación aritmética es muy pequeño, pero no siempre es así.

Después de realizar varias operaciones con varios números, es posible que el resultado r' no sea el número flotante r^* que mejor represente al resultado exacto r. O sea, que $r' \neq r^* = fl(r)$. Esto es debido a que algunas operaciones amplifican los errores que se acumulan por varias otras operaciones. Debido a lo anterior, es necesario contar con alguna medida de la validez o significancia de los resultados de las operaciones. Dicha medida son los dígitos significativos. Se dice que r' **representa a** r **con** q **dígitos significativos** si

$$\frac{|r' - r|}{r} < \frac{b^{-q}}{2}$$

Lo ideal es que $q = p$ o $q = p - 1$ pero no siempre es así. En algunas operaciones específicas, como el ejemplo siguiente, se pierden muchos dígitos significativos.

Una fuente común de errores es restar dos números muy similares, debido a que varios de sus principales dígitos son iguales y entonces hay pocos dígitos distintos para calcular la diferencia. De esta manera se pierden dígitos significativos. Un ejemplo es cuando en la fórmula para calcular una de las raíces cuadradas

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

b^2 y $4ac$ son muy similares. A veces es posible evitar dichas restas utilizando fórmulas alternativas para calcular el mismo valor. En el caso de la raíz cuadrada, la fórmula es

$$x_1 = \frac{-2c}{b + \sqrt{b^2 - 4ac}}$$

C.2.2. Implementación de los números flotantes

El estándar IEEE 754 para la aritmética en punto flotante [21](en sus versiones de 1985 y 2008) define la representación de números flotantes en la computadoras actuales.

De especial interés son las especificaciones de los **números flotantes de 32 y 64 bits**, que son los más usados.

- Al formato de 32 bits se le llama *binary32*. El signo se codifica en el primer bit, la característica en $p = 24$ bits (el primer bit es implícito, así que no se guarda) y el exponente en $l = 8$ bits, lo que permite representar exponentes entre -128 y 127.
- Al formato de 64 bits se le llama *binary64*. El signo se codifica en el primer bit, la característica en 53 bits (el primer bit es implícito) y el exponente en 11 bits, para representar exponentes entre -1024 y 1023.

La posibilidad de usar un bit implícito se debe a que se usan números normalizados. Como en un número normalizado se sabe que la parte entera vale cero, no es necesario codificarlo explícitamente. Su uso permite tener un mayor rango de números representables. A la hora de realizar operaciones, este bit se incluye en la codificación del número.

Apéndice D

Generación de números aleatorios

Todos los métodos de Montecarlo tienen en común que trabajan de manera probabilística, así que necesitan de una fuente de números (pseudo)aleatorios para realizar cálculos. Los más utilizados son generadores con una distribución uniforme de probabilidad en el intervalo (0, 1). A partir de ellos es posible generar números en cualquier otro intervalo y con otras distribuciones de probabilidad.

Hay dos maneras de obtener números aleatorios: de algún proceso físico o a partir de un algoritmo diseñado específicamente para generar números pseudoaleatorios. La mayoría de las simulaciones de Montecarlo utiliza estos algoritmos, pero se hablará brevemente de los métodos físicos.

Además es necesario verificar que las secuencias de números producidos parezcan genuinamente aleatorios. Para ello existen pruebas de calidad para verificar que ello se cumple.

D.1. Fuentes físicas de números aleatorios

Hay diversos fenómenos físicos que ocurren de manera aleatoria. La naturaleza aleatoria de de estos fenómenos puede ser intrínseca (como la mecánica cuántica) o debida a un conocimiento incompleto del estado del sistema físico (como tirar dados). Estos fenómenos se pueden utilizar como fuentes de números aleatorios. Algunas de las fuentes son:

- Tirar dados
- Ruleta eléctrica [25]
- Una computadora (movimientos de ratón o teclado, ruido del disco duro) [28]
- El decaimiento radiactivo de algunos elementos [26]
- Ruido atmosférico [24]
- Láseres [22, 23]

A excepción de ciertos generadores basados en láseres, el resto de los generadores físicos generan números aleatorios de manera lenta comparado con lo que se necesitan para realizar simulaciones. Por esto, las simulaciones de Montecarlo obtienen los números aleatorios que necesita de generadores de números pseudoaleatorios.

D.2. Generadores de números pseudoaleatorios

La otra manera de generar números aleatorios es a través de algoritmos conocidos como **generadores de números pseudoaleatorios** (PRNG, *pseudo random number generators* en inglés) [1]. Estos números no son genuinamente aleatorios, pero pueden parecerlo lo suficiente para varios propósitos. A diferencia de los métodos físicos, estos algoritmos pueden generar números a una gran velocidad.

Varios de los generadores proporcionan números enteros entre 0 y un valor máximo $m-1$ que después pueden ser convertidos a números flotantes. La forma más sencilla de estos generadores es

$$i_n = f(i_{n-1})$$

donde f es una función que solamente utiliza aritmética de números enteros: suma, multiplicación o división entera. La función requiere de un número inicial i_0, denominado **semilla**. Cada semilla puede generar una sucesión distinta de números; también se puede repetir la misma sucesión utilizando la misma semilla, lo que puede ser muy útil a la hora de depurar algoritmos de Montecarlo.

Debido a la estructura de los algoritmos, las secuencias que generan son periódicas, que a su vez implica secuencias con un número limitado de números. Por lo tanto, un buen generador debe poseer un periodo k largo. El periodo máximo posible es m, el total de números enteros del rango $[0, m-1]$ del generador.

Otros generadores tienen la forma

$$i_n = f(i_{n-1}, i_{n-2}, \ldots, i_{n-q})$$

por lo que requieren más semillas y el periodo máximo que pueden tener es m^q.

A continuación se hablará brevemente de algunos tipos de generadores.

D.2.1. Generador lineal congruencial

Uno de los primeros y más utilizados tipos de generadores de números aleatorios es el **generador lineal congruencial** o GLC (LGC, *linear congruential generator* en inglés). El generador lineal congruencial tiene la forma

$$i_n = (ai_{n-1} + c) \bmod m$$

donde $a \bmod b$ representa la operación módulo.

Se han hecho muchas investigaciones acerca de cuáles números a, c, m e i_0 son los adecuados para obtener un GLC con las siguientes características:

- Periodo largo: el máximo periodo posible es el entero más grande que soporte la computadora. En una computadora de 32 o 64 bits, el tamaño máximo es $w = 2^{32} \approx 4.3 \times 10^9$ o $2^{64} \approx 1.8 \times 10^{19}$, aunque hay lenguajes limitados a tamaños menores.
- Que genere secuencias con suficiente apariencia de aleatoriedad
- Rápido. Se han investigado varias maneras elegir constantes y reducir el número y complejidad de las operaciones para aumentar la velocidad de los algoritmos.

La solución más aceptada para implementar el GLC es el método o truco de Schrage, el cual multiplica dos números de 32 o 64 bits módulo otro número de 32 (64) bits, con lo que se obtiene un generador con periodo $m = w - 1$.

D.2.2. Reordenamiento de Bays-Durham

Un problema del GLC es que genera números con algunas correlaciones. Una de ellas es que si de una secuencia larga se toman números en grupos de q elementos y se consideran como puntos en un espacio q-dimensional, estos puntos se agrupan en planos en vez de estar distribuidos uniformemente.

El método de reordenamiento de Bays-Durham (*Bays-Durham shuffling*) se utiliza para eliminar dichas correlaciones (al menos a bajas dimensiones) y además aumenta el periodo del generador considerablemente. Consiste en primero generar una lista de N números enteros $\{j_n\}$ con el generador lineal congruencial. Usualmente se considera $N \approx 100$. Luego se genera un número extra, denominado y. Después se repite el siguiente algoritmo:

1. Se calcula $k = \lfloor yN/m \rfloor$, donde $\lfloor x \rfloor$ es la función piso, que regresa el entero más grande menor o igual a x. La definición se hace tal que $0 \leq k \leq N - 1$.

2. Se escoge el elemento j_k de la lista como el nuevo número aleatorio obtenido del generador y se asigna $y = j_k$.

3. Se genera un nuevo número con el GLC y se asigna a j_k.

El uso combinado del GLC con el método de Bays-Durham genera secuencias con un periodo de hasta m^N y elimina las correlaciones que se obtienen con el uso exclusivo del generador lineal congruencial.

D.2.3. Mersenne Twister

Existe una larga lista de otros generadores. Sin embargo, actualmente uno de los más populares es el Mersenne Twister [29, 30], que es el generador de números pseudoaleatorios por defecto de los lenguajes Python y Julia (los lenguajes que se usaron para las simulaciones de este trabajo) y en otros paquetes de software, por lo que es importante su mención.

El generador Mersenne Twister posee un periodo k de $2^{19937} - 1 \approx 4.3 \times 10^{6001}$ *, independientemente de si la implementación es para 32 o 64 bits y se distribuye uniformemente hasta en 623 dimensiones. A diferencia del generador lineal congruencial, que genera números enteros, Mersenne Twister genera por sí mismo números reales (o flotantes) en el intervalo [0, 1]. Según los autores, este algoritmo es adecuado para simulaciones de Montecarlo. La teoría usada para desarrollar este generador queda fuera del alcance de este trabajo.

D.3. Calidad de los generadores de números pseudoaleatorios

Existen diversos métodos para comparar las propiedades de los generadores en sí mismos o a las secuencias que producen con las propiedades de números generados con la distribución uniformemente aleatoria. En esta sección solo se hablará de las pruebas realizadas a secuencias (pruebas empíricas) y no a los algoritmos en sí mismos.

* El valor de k en base decimal se puede calcular directamente en la consola de Python con la instrucción `2**19937 - 1`, donde ** es la operación de potencia. Esto es posible porque Python representa enteros muy grandes a nivel de software para evitar las limitaciones impuestas por el hardware. Los primeros tres dígitos son 431. El número de dígitos necesarios para representar k en base 10 se obtiene con la instrucción `len(str(k))`, que devuelve un resultado de 6002. Entonces, en notación científica, hay 6001 dígitos después del punto decimal.

D.3.1. Pruebas empíricas

Las pruebas empíricas para números (pseudo)aleatorios [18] son aquellas que utilizan una secuencia de números obtenida de algún generador para cuantificar su calidad. Estas pruebas comparan alguna propiedad de la secuencia a estudiar con la que poseería una secuencia aleatoria. Se han propuesto varios conjuntos de pruebas empíricas. El primer conjunto lo propuso Knuth en 1968. Otros conjuntos de pruebas conocidas son las pruebas Diehard (Marsaglia, 1985, 1995), las del NIST (2000) y TestU01 (L'Ecuyer, 1998, 2007).

Algunas de las pruebas más sencillas, descritas por Knuth e incluidas en varios de estos conjuntos de pruebas son las siguientes:

- Equidistribución o frecuencia. Se calcula si la secuencia está distribuida uniformemente en [0, 1).
- Prueba serial. Esta prueba calcula si números organizados en pares (o de manera menos común, en n-adas más grandes) están distribuidos uniformemente en un espacio multidimensional.
- Prueba de huecos (*gap test*). La prueba mide los "huecos" entre dos números U_j y U_{j+r}. A su vez, $U_j, U_{j+r} \in [\alpha, \beta]$.
- Prueba de póker. En grupos de 5 números, se cuenta cuántas veces ocurre que hay r números distintos, $r = 1, 2, 3, 4, 5$. Por ejemplo, en (3, 64, 22, 64, 35) hay $r = 4$ números distintos.
- Prueba del coleccionista de cupones.
- Permutaciones.
- *Run test*. Se estudia la longitud de subsecuencias de números crecientes o decrecientes.
- Máximo de t.
- Prueba de colisión.
- Correlación serial.

Varias de las pruebas mencionadas recurren a una o a las dos siguientes pruebas para conocer la probabilidad de que la secuencia estudiada haya sido generada aleatoriamente: la **prueba de chi cuadrada** y la **prueba de Kolmogórov-Smirnov**.

D.3.2. Prueba de chi cuadrada (χ^2)

Esta prueba se aplica para observaciones independientes que se pueden clasificar en un conjunto de k categorías. Consiste en averiguar la probabilidad p de que en una secuencia de n eventos se generen Y_s observaciones de la categoría s. El total de observaciones de cada categoría se compara con una distribución de probabilidad teórica esperada, en la cual cada categoría tiene una probabilidad p_s. De esta manera, el número teórico de observaciones es aproximadamente np_s. Se comparan Y_s y np_s utilizando la fórmula

$$V = \sum_{s=1}^{k} \frac{(Y_s - np_s)^2}{np_s}$$

Después se calcula la probabilidad p para que una secuencia aleatoria con k categorías (o específicamente, para $\nu = k-1$ grados de libertad) genere un valor menor o igual a V. La probabilidad p se obtiene de la función de distribución acumulada chi cuadrada (χ^2 o de Pearson)

$$F_\nu(x) = \frac{\gamma(\nu/2, x/2)}{\Gamma(\nu/2)}$$

donde γ y Γ son las funciones gamma incompletas inferior y superior, respectivamente.

$$\gamma(a, x) = \int_0^x t^{a-1} e^{-x} dt$$

$$\Gamma(a, x) = \int_x^\infty t^{a-1} e^{-x} dt$$

De esta manera, $p = F_\nu(x = V)$.

Existen tablas organizadas con varios valores de ν, p y V, pero actualmente se puede obtener p con más facilidad y precisión utilizando software especializado en estadística, incluso evitando al usuario el cálculo explícito de V.

Una vez calculada p, se decide si la secuencia pudo ser generada o no por la distribución de probabilidad propuesta. Las reglas están descritas más adelante.

D.3.3. Prueba de Kolmogórov–Smirnov o prueba KS

A diferencia de la prueba de chi cuadrada, la prueba de Kolmogórov–Smirnov se aplica para secuencias de números que pueden adquirir un número infinito de valores (distribuciones continuas), como una secuencia aleatoria de números reales. Si de la variable X se obtiene la secuencia $X_1, X_2, \ldots, X_n$, se define la distribución cumulativa de probabilidad empírica como

$$F_n(x) = \frac{\text{número de } X_1, X_2, \ldots, X_n \text{ tal que } \leq x}{n}$$

En esta prueba se comparan las diferencias máximas de $F_n(x)$ por arriba y por abajo de la distribución esperada $F(x)$, la cual es una función continua. Para ello se calculan

$$K_n^+ = \sqrt{n} \max_{-\infty < x < \infty} (F_n(x) - F(x))$$

$$K_n^- = \sqrt{n} \max_{-\infty < x < \infty} (F(x) - F_n(x))$$

Las probabilidades de que $F(x)$ genere valores menores o iguales a K_n^+ y K_n^- se encuentran en tablas o se pueden calcular con software de estadística.

Para $0 \leq t \leq n$, la probabilidad de que $K_n^+ \leq \frac{t}{\sqrt{n}}$ o $K_n - + \leq \frac{t}{\sqrt{n}}$ es

$$p = \frac{t}{n^n} \sum_{k=0}^{t} \binom{n}{k} (k-t)^k (t+n-k)^{n-k-1}$$

Para decidir si la secuencia es lo suficientemente aleatoria, se utilizan las siguientes reglas.

D.3.4. Reglas para interpretar p

Al encontrar p con cualquiera de los métodos ya descritos, se decide si la secuencia es lo suficientemente aleatoria. Si $p < 0.01$ o $p > 0.99$, la secuencia es muy poco aleatoria (se ajusta muy poco a las probabilidades esperadas o es "demasiado buena para ser cierta") y se rechaza la hipótesis de que hayan sido generados con la distribución esperada propuesta. Para los demás casos se acepta la hipótesis de que los números han sido generados por la distribución propuesta, aunque dependiendo del caso, se hace con reservas. Si $0.01 < p < 0.05$ o $0.95 < p < 0.99$, es poco probable. Si $0.05 < p < 0.1$ o $0.9 < p < 0.95$, es probable. Finalmente, si $0.1 < p < 0.9$, es casi seguro que sí hayan sido generados con las probabilidades propuestas.

Apéndice E

Códigos de las simulaciones

A continuación se muestran los programas que se desarrollaron para realizar las simulaciones de los sistemas estudiados.

E.1. Programa del modelo L en el ensamble canónico

```
!Este programa solo permite cambiar de quiralidad de manera
!horizontal y vertical, SIN rotaciones de 180°

!Se sustituyó el guion bajo con diagonal-guion bajo para que salgan bien los comentarios

program tesis\_canonico
implicit none

!********** DEFINICION DE VARIABLES *********
!********** PARAMETROS DEL SISTEMA *********
!Estos parámetros se pueden dejar en otro archivo de lectura
!Los valores de lambda también podrían dejarse en otro archivo de lectura
integer, parameter :: side = 50, nsample = side**2, cycles = side**5, nmol = side**2,
    ntemps = side
real(8), parameter :: l1 = 0.9, l2 = 0.45, init\_temperature = 3.0, final\_temperature
     = 0.3
! En caso de que se quiera modificar o no, pongo el rango de interaccion aparte
real(8) :: lambda

integer :: nps, neigh\_number
integer :: i\_lambda = 1, f\_lambda = 20

!********** MATRICES A USAR *********
real(8), dimension(nmol, 2) :: mol\_centers
real(8), dimension(nmol, 3, 2) :: positions
integer, dimension(nmol) :: chirality, orientation
integer, allocatable, dimension(:, :) :: neighbors
real(8), allocatable, dimension(:, :, :) :: neighbor\_pos
real(8), dimension(cycles/nsample + 2) :: system\_energy
```

```
28
29 !********** VARIABLES INTERNAS *********
30 integer :: ii, jj, kk, sample\_counter, counter, lambda\_iter, size\_seed, cycles\
       _ptemp, temp\_counter
31 real(8) :: energy\_change, energy, size\_sys, eps, zero, pi, temperature, dtemp, quad\
       _a, quad\_b
32 real(8), dimension(ntemps + 1) :: temp\_arr
33 integer :: new\_chirality
34 real(8), dimension(3, 2) :: new\_pos
35 character(1000) :: chir\_format, filename, filename2
36 character(1000) :: directory
37 real(8), dimension (4, 2) :: cell
38 real(8), dimension(32) :: relative\_dist
39 real(8), dimension(2) :: temp\_center
40 real(8), dimension(20) :: unique\_dist !De aquí se obtienen las lambda
41 integer, dimension(8) :: values
42 integer, dimension(:), allocatable :: seed
43
44
45 !Directorio donde se guardarán los resultados
46 directory = "lado50\_tiempo5\_enfr/temp03/"
47 print*, trim(directory)
48
49 !eps es el valor que le sumaba a lambda para evitar errores de redondeo
50 !pero parece que no es necesario
51 eps = 0.0
52
53 !Formato de la matriz dependiente del tamaño
54 !side sustituye a I2
55 !Es para la escritura de la matriz del sistema
56 write(chir\_format, '( "(", I2, "I4)" )') side
57
58 !Cálculo de los valores posibles de lambda tal que alcancen todos los sitios
59 !de interacción de las moléculas primeras vecinas
60
61 !Estas son las posible posiciones de los sitios de interacción de una molécula
62 !Los asteriscos son los sitios de interacción
63 ! 2 * * 4
64 ! —
65 ! —
66 ! —
67 ! l1 —
68 ! —
69 ! —
70 ! —
71 ! 1 *————* 3
72 ! (0,0) l2
73
74 zero = 0.0 !Es tipo real(8)
75  cell(1,:) = (/zero, zero/)
76  cell(2,:) = (/zero, l1/)
77  cell(3,:) = (/l2, zero/)
```

```
 cell(4,:) = (/l2, l1/)

!Las posiciones de los sitios de interacción de las moléculas más cercanas
!se encuentran desplazadas en los ejes x, y
!Aquí los asteriscos son los centros de red donde se localizan las moléculas vecinas.
!En cada una está centrada una molécula como la de arriba.
! * * *
! (-1,1) (1,0) (1,1)


! * * *
! (-1,0) (0,0) (1,0)


! * * *
! (-1,-1) (-1,0) (-1,1)

!Con la información de arriba, se calculan las 8*4 = 32 posibles distancias entre
!sitios de interacción de moléculas vecinas
counter = 1
do jj = -1, 1 !Desplazamiento en el eje y
    do ii = -1, 1 !Desplazamiento en el eje x
        do kk = 1, 4 !Número de sitios de interacción por molécula
            if (ii == 0 .and. jj == 0) then !La molécula central no se incluye
                cycle
            end if
            temp\_center = (/dble(ii), dble(jj)/)
            relative\_dist(counter) = norm(cell(kk, :) + temp\_center)
            counter = counter + 1
        end do
    end do
end do

!write(*,*) "Distancias"
!write(*, "(F20.18)") relative_dist

!Como hay distancias repetidas, hay que ver cuáles son únicas
!Hay 20 distancias únicas
unique\_dist = 0.0
counter = 1
do ii = 1, 32
    if (.not.(any(abs(unique\_dist - relative\_dist(ii)) < 0.00001))) then
! if (.not.(any(unique_dist == relative_dist(ii)))) then
        unique\_dist(counter) = relative\_dist(ii)
        counter = counter + 1
    end if
end do

!Cambiar semilla del PRNG con la fecha y hora del momento de ejecución
!A ver si así cambia la salida
call date\_and\_time(VALUES=values)
```

```
129 call random\_seed(size=size\_seed)
130 allocate(seed(size\_seed))
131 seed(:) = values(:)
132 call random\_seed(put=seed)
133
134
135 !write(*,*) "Distancias únicas"
136 !write(*, "(F20.18)") unique_dist
137
138 !Se realiza una simulación por cada posible distancia entre sitios de interacción
139 !Los valores de lambda son precisamente las posibles distancias
140
141 do lambda\_iter = i\_lambda, f\_lambda
142     lambda = unique\_dist(lambda\_iter) + eps
143     write(*,*) "lambda = ", lambda
144
145     !Número de vecinos efectivos por "lado"
146     !Es para reducir la búsqueda de vecinos
147     !nps: neighbors per side
148     if (lambda <= 1.005) then
149         nps = 1
150     else
151         nps = 2
152     end if
153     !si nps = 1, entonces neigh_number = 8
154 ! * * *
155 ! * 0 *
156 ! * * *
157
158     !si nps = 2, entonces neigh_number = 24
159 ! * * * * *
160 ! * * * * *
161 ! * * 0 * *
162 ! * * * * *
163 ! * * * * *
164
165     neigh\_number = (2*nps + 1)**2 - 1
166
167     !Asignación de memoria después de calcular neigh_number
168     allocate(neighbors(nmol, neigh\_number))
169     allocate(neighbor\_pos(neigh\_number + 1, 3, 2))
170
171     !Inicialización
172     call create\_molecules(nmol, mol\_centers, positions, chirality, orientation,
            neighbors)
173
174
175     !Energía inicial
176     energy = 0.0
177     do jj = 1, nmol
178         ! Posición de la molécula central"
```

```
179         neighbor\_pos(1, :, :) = positions(jj, :, :)
180         ! Se buscan las posiciones de las vecinas
181         do ii = 1, neigh\_number
182             neighbor\_pos(ii + 1, :, :) = positions(neighbors(jj, ii), :, :)
183         end do
184         energy = energy + potential\_energy(neighbor\_pos)
185     end do
186     energy = energy / 2.0
187     size\_sys = dble(side*side)
188     system\_energy(1) = energy/size\_sys
189 
190     ! Ciclo de Montecarlo
191     ! Se modificó para cambiar la temperatura poco a poco
192     dtemp = (init\_temperature - final\_temperature)/(ntemps - 1)
193 
194     !Quiero convertir el dt lineal a uno cuadrático, para que la temperatura descienda rapido
195     !al principio y lento al final, a ver si funciona para correr con side**4
196     quad\_a = (init\_temperature - final\_temperature)/(init\_temperature**2 - final\
            _temperature**2)
197     quad\_b = final\_temperature - quad\_a*final\_temperature**2
198     temp\_arr(1) = init\_temperature
199     do ii = 1, (ntemps - 2)
200         temp\_arr(ii + 1) = quad\_a*(init\_temperature - ii*dtemp)**2 + quad\_b
201     end do
202     temp\_arr(ntemps) = final\_temperature
203     temp\_arr(ntemps + 1) = final\_temperature
204 
205     write(*, *) temp\_arr
206     temperature = init\_temperature
207     !Número de ciclos que se hacen por cada temperatura
208     cycles\_ptemp = floor(dble(cycles)/ntemps)
209 
210     sample\_counter = 1
211     temp\_counter = 1
212     do ii = 1, cycles
213         call montecarlo(nmol, mol\_centers, positions, chirality, orientation, &
214             neighbors, temperature, energy\_change)
215         energy = energy + energy\_change
216         !Muestreo
217         if (mod(ii, nsample) == 0) then
218             system\_energy(sample\_counter) = energy/size\_sys
219             sample\_counter = sample\_counter + 1
220             !if (ii == cycles/10) then
221             ! write(*,*) "Avance: ", 100*ii/cycles, iend if
222         end if
223         !Bajar la temperatura cada cierto número de ciclos
224         if (mod(ii, cycles\_ptemp) == 0) then
225 ! print*, "Temp. intermedia: ", temperature
226             temp\_counter = temp\_counter + 1
227 ! print*, temp_counter
228             temperature = temp\_arr(temp\_counter)
229         end if
```

```
230
231     end do
232
233     ! Escritura de archivos
234     ! La siguiente línea es para variar el nombre del archivo conforme a lambda
235     ! Archivo de la configuración del sistema
236     ! Aquí es donde se debería multiplicar chirality*orientation para
237     ! tener en un solo número la quiralidad y la orientación
238     write(filename, '( A, "lambda", F7.5, "\_final.txt" )') trim(directory), lambda
239     print*, trim(filename)
240     open (10, file = filename, status='UNKNOWN')
241     write(10, chir\_format) chirality*orientation
242     close(10)
243
244     !Archivo de las energías
245     write(filename2, '( A, "t\_energies\_lambda", F7.5, ".txt" )') trim(directory),
            lambda
246     open(20, file = filename2, status='UNKNOWN')
247     write(20, *) system\_energy
248     close(20)
249
250     !Deasignación de tamaño, para asignarle otra dependiendo de nps
251     deallocate(neighbors)
252     deallocate(neighbor\_pos)
253 end do
254
255 contains
256
257 !********** FUNCIONES Y SUBRUTINAS *****************
258
259 !Norma de un vector
260 !En el argumento iría la resta de dos vectores si es necesario
261 real(8) function norm(vr)
262     implicit none
263
264     real(8), dimension(2) :: vr
265     norm = sqrt(vr(1)**2 + vr(2)**2)
266
267 end function norm
268
269 !Elegir un número entero entre a y b, incluidos a y b
270 integer function rand\_int(a, b)
271     implicit none
272     integer, intent(in) :: a, b ! a ¡b
273     integer :: diff
274     real(8) :: rand
275
276     diff = dble(b + 1 - a)
277     call RANDOM\_NUMBER(rand)
278     rand\_int = floor(diff*rand) + a !El + 1.0 es para que tambien seleccione b
279
280 end function rand\_int
```

```
281
282 !Pozo cuadrado
283 !El argumento es un vector
284 real(8) function square\_well(r\_ij)
285     implicit none
286
287     real(8), dimension(2) :: r\_ij
288     real(8) :: ans, dist
289     dist = norm(r\_ij)
290     if (dist == 0.0) then
291         ans = 1000.0 !Es para simular infinitos, tal vez me traiga problemitas
292     else if (dist > 0 .and. dist <= lambda) then
293         ans = -1.0
294     else !dist ¿lambda
295         ans = 0.0
296     end if
297     square\_well = ans
298 end function square\_well
299
300
301 !Moléculas derechas
302 !Quiero que la función me regrese una matriz
303 !real(8) function dextro_molecule(square_pos) NO FUNCIONA
304 !real(8), dimension(3, 2) function dextro_molecule(square_pos) NO FUNCIONA
305 function dextro\_molecule(square\_pos, orientation)
306 ! Regresa una lista con las posiciones de los átomos de cada molécula para que sea derecha
307     implicit none
308     real(8), dimension(3, 2) :: pos, dextro\_molecule !Definí el tipo de la función
            ADENTRO, no antes del nombre
309     real(8), dimension(2) :: square\_pos
310     integer :: orientation, ii
311                     ! 3
312     pos(1, 1) = -l2/2.0    ! *
313     pos(1, 2) = -l1/2.0    ! —
314     pos(2, 1) = l2/2.0    ! —
315     pos(2, 2) = -l1/2.0    ! —
316     pos(3, 1) = l2/2.0    ! —
317     pos(3, 2) = l1/2.0    ! *———*
318                     ! 1 2
319
320     do ii = 1, 3
321         if (orientation == 1) then
322             pos(ii, :) = square\_pos(:) + pos(ii, :)
323         else if (orientation == 2) then
324             ! El signo menos basta para voltearla 180 grados
325             pos(ii, :) = square\_pos(:) - pos(ii, :)
326         else
327             print*, "ERROR DERECHAS"
328         end if
329     end do
330     !La siguiente línea era lo que me faltaba :v
331     dextro\_molecule = pos
```

```
332 end function dextro\_molecule
333
334 !Moléculas izquierdas
335 function levo\_molecule(square\_pos, orientation)
336 ! Regresa una lista con las posiciones de los átomos de cada molécula para que sea derecha
337     implicit none
338     real(8), dimension(3, 2) :: pos, levo\_molecule !Definí el tipo de la función ADENTRO,
            no antes del nombre
339     real(8), dimension(2) :: square\_pos
340     integer :: orientation, ii
341                     ! 3
342     pos(1, 1) = l2/2.0    ! *
343     pos(1, 2) = -l1/2.0    ! —
344     pos(2, 1) = -l2/2.0    ! —
345     pos(2, 2) = -l1/2.0    ! —
346     pos(3, 1) = -l2/2.0    ! —
347     pos(3, 2) = l1/2.0    ! *————*
348                     ! 2 1
349     do ii = 1, 3
350         if (orientation == 1) then
351             pos(ii, :) = square\_pos(:) + pos(ii, :)
352         else if (orientation == 2) then
353             pos(ii, :) = square\_pos(:) - pos(ii, :)
354         else
355             print*, "ERROR IZQUIERDAS"
356         end if
357     end do
358
359     levo\_molecule = pos
360 end function levo\_molecule
361
362 subroutine create\_molecules(nmol, mol\_centers, positions, chirality, orientation,
        neighbors)
363     implicit none
364     integer, intent(in) :: nmol
365     real(8), dimension(nmol, 2), intent(out) :: mol\_centers
366     real(8), dimension(nmol, 3, 2), intent(out) :: positions
367     integer, dimension(nmol), intent(out) :: chirality, orientation
368     integer, dimension(nmol, neigh\_number), intent(out) :: neighbors
369     real(8) :: rand
370     integer :: ii, jj, row, col, mol\_index, counter
371
372     ! Hacer que "la mitad" de las moléculas tengan la misma quiralidad de manera aleatoria
373     do ii = 1, nmol
374         call RANDOM\_NUMBER(rand)
375         if (rand < 0.5) then
376             chirality(ii) = 1 !"D"
377         else
378             chirality(ii) = -1 !"L"
379         end if
380
381         ! Creo que es mejor así y no con rand_int para decir cuál
```

```
382         ! orientación es cuál
383         call RANDOM\_NUMBER(rand)
384         if (rand < 0.5) then
385             orientation(ii) = 1 !Lado corto abajo
386         else
387             orientation(ii) = 2 !Lado corto arriba
388         end if
389     end do
390 
391     !chirality = 1
392 
393     ! Sitúa las moléculas en posiciones y orientaciones aleatorias
394     do jj = 0, (side - 1)
395         do ii = 0, (side - 1)
396             mol\_index = jj*side + ii + 1
397             mol\_centers(mol\_index, 1) = ii
398             mol\_centers(mol\_index, 2) = jj
399             if (chirality(mol\_index) == -1) then
400                 !temp_pos = levo_molecule(mol_center(mol_index, :))
401                 positions(mol\_index, :, :) = &
402                 levo\_molecule(mol\_centers(mol\_index, :), orientation(mol\_index))
403             else if (chirality(mol\_index) == 1) then
404             !temp_pos = dextro_molecule(mol_center(mol_index, :))
405                 positions(mol\_index, :, :) = &
406                 dextro\_molecule(mol\_centers(mol\_index, :), orientation(mol\_index))
407             end if
408             !Asignar vecinos
409             !nps está definido desde arriba
410             counter = 1
411             do col = -nps, nps
412                 do row = -nps, nps
413                     if (col == 0 .and. row == 0) then
414                         cycle  !Se salta la molécula central porque no es su propia vecina
415                     else
416                         neighbors(mol\_index, counter) = modulo(jj + col, side)*side +
                                modulo(ii + row, side) + 1
417                         counter = counter + 1
418                     end if
419                 end do
420             end do
421         end do
422     end do
423 end subroutine create\_molecules
424 
425 ! Esta función selecciona la quiralidad contraria y elige al azar una orientación,
426 ! que es equivalente a elegir al azar voltear horizontal o verticalmente
427 ! Para 4 posibles rotaciones tal vez deba cambiar el programa
428 subroutine change\_chirality(mol\_chirality, center, new\_chirality, new\_pos, new\
        _orientation)
429 implicit none
430 integer, intent(in) :: mol\_chirality
431 real(8), dimension(2), intent(in) :: center
```

```
integer, intent(out) :: new\_chirality, new\_orientation
real(8), dimension(3, 2), intent(out) :: new\_pos
real(8) :: rand

    ! Se escoge al azar si se voltea sobre
    ! el lado largo o el corto
    new\_orientation = rand\_int(1, 2)

    !Si es derecha, cambiar a izquierda
    if (mol\_chirality == -1) then
        new\_pos = dextro\_molecule(center, new\_orientation)
    else if (mol\_chirality == 1) then
        new\_pos = levo\_molecule(center, new\_orientation)
    end if
    new\_chirality = -mol\_chirality

end subroutine change\_chirality

!Esta función rota 180 grados una particula, sin cambiar su quiralidad
subroutine rotate\_molecule(mol\_chirality, center, old\_orientation, new\_pos, new\
    _orientation)
    integer, intent(in) :: mol\_chirality, old\_orientation
    real(8), dimension(2), intent(in) :: center
    real(8), dimension(3, 2), intent(out) :: new\_pos
    integer, intent(out) :: new\_orientation

    !Conserva quiralidad, solo rota
    if (mol\_chirality == 1) then
        if (old\_orientation == 1) then
            new\_orientation = 2
            new\_pos = dextro\_molecule(center, new\_orientation)
        else if (old\_orientation == 2) then
            new\_orientation = 1
            new\_pos = dextro\_molecule(center, new\_orientation)
        end if
    else if (mol\_chirality == -1) then
        if (old\_orientation == 1) then
            new\_orientation = 2
            new\_pos = levo\_molecule(center, new\_orientation)
        else if (old\_orientation == 2) then
            new\_orientation = 1
            new\_pos = levo\_molecule(center, new\_orientation)
        end if
    end if
end subroutine rotate\_molecule

real(8) function potential\_energy(positions)
    implicit none
    ! positions incluye las posiciones de cada átomo
    ! positions[1] es la molécula central

    real(8), dimension(neigh\_number + 1, 3, 2) :: positions
```

```
483     integer :: ii, jj, kk, aa
484     real(8) :: energy, temp\_dist
485     real(8), dimension(2) :: dist
486 
487     energy = 0.0
488     do kk = 2, (neigh\_number + 1) !Iterar sobre las partículas vecinas
489         do jj = 1, 3 ! Sobre los sitios atractivos de las vecinas
490             do ii = 1, 3 !Sobre las de la partícula central
491                 ! Sin condiciones periódicas
492                 !energy += square_well(positions[1][ii] - positions[kk][jj])
493 
494                     !Con condiciones periódicas
495                 do aa = 1, 2 !Coordenadas de los sitios
496                     temp\_dist = abs(positions(1, ii, aa) - positions(kk, jj, aa))
497                     if (temp\_dist > side/2) then
498                         temp\_dist = side - temp\_dist
499                     end if
500                     dist(aa) = temp\_dist
501                 end do
502                 energy = energy + square\_well(dist)
503             end do
504         end do
505     end do
506     potential\_energy = energy
507 end function potential\_energy
508 
509 !Criterio de Metropolis
510 logical function metropolis\_step(exp\_arg)
511 !Returns True if the trial configuration goes towards a region with higher probability
512 !or it is given the chance to explore regions with fewer probability
513 
514 implicit none
515 
516 real(8) :: exp\_arg, w, rand
517 logical :: accepted
518 
519     if (exp\_arg < 0.0) then
520         accepted = .true. ! Se actualiza el estado del sistema
521     else
522         w = exp(-exp\_arg)
523         call RANDOM\_NUMBER(rand)
524         if (rand < w) then ! Energía: w
525             accepted = .true. ! También se actualiza el estado del sistema
526     else
527         accepted = .false.
528         end if
529     end if
530     metropolis\_step = accepted
531 
532 end function metropolis\_step
533 
```

```
534 subroutine montecarlo(nmol, centers, positions, chirality, orientation, neighbors, t\
        _env, energy\_change)
535     implicit none
536     ! Variables de entrada y salida
537     integer, intent(in) :: nmol
538     real(8), dimension(nmol, 2), intent(in) :: centers
539     integer, dimension(nmol, neigh\_number), intent(in) :: neighbors
540     real(8), dimension(nmol, 3, 2), intent(inout) :: positions
541     integer, dimension(nmol), intent(inout) :: chirality, orientation
542     real(8), intent(in) :: t\_env
543     real(8), intent(out) :: energy\_change
544 
545     !Variables internas
546     integer :: ii, idx, neighbor, trial\_chir
547     real(8), dimension(neigh\_number + 1, 3, 2) :: neighbor\_pos, neighbor\_pos2
548     real(8), dimension(3, 2) :: trial\_pos
549     real(8) :: old\_energy, trial\_energy, energy\_diff, rand
550     integer :: trial\_orientation
551     logical :: accepted
552 
553     ! Elegir una molécula al azar
554     idx = rand\_int(1, side*side)
555     ! Buscar las posiciones de los átomos de esa molécula y de sus vecinas
556     neighbor\_pos(1, :, :) = positions(idx, :, :)
557     !write(*,*) "Posición: ", positions(idx, :, :)
558     do ii = 1, neigh\_number
559         neighbor = neighbors(idx, ii)
560         neighbor\_pos(ii + 1, :, :) = positions(neighbor, :, :)
561     end do
562 
563     ! Intento de cambio
564     ! En este caso, puede elegir entre cambiar quiralidad o rotar
565     call random\_number(rand)
566     if (rand < 0.5) then
567         call change\_chirality(chirality(idx), centers(idx, :), trial\_chir, &
568         trial\_pos, trial\_orientation)
569     else
570         call rotate\_molecule(chirality(idx), centers(idx, :), orientation(idx), &
571         trial\_pos, trial\_orientation)
572         trial\_chir = chirality(idx)
573     end if
574 
575     ! Buscar las posiciones de los átomos de esa molécula y de sus vecinas
576     neighbor\_pos2(1, :, :) = trial\_pos
577     do ii = 1, neigh\_number
578         neighbor = neighbors(idx, ii)
579         neighbor\_pos2(ii + 1, :, :) = positions(neighbor, :, :)
580     end do
581 
582     ! Cálculo de energías
583     old\_energy = potential\_energy(neighbor\_pos)
584     trial\_energy = potential\_energy(neighbor\_pos2)
```

```
585
586     ! La diferencia de energías se utiliza para aceptar o rechazar el movimiento.
587     ! energy_change entrega la diferencia de energías si el movimiento fue aceptado.
588     ! Si fue rechazado, no hay cambio de energía en el sistema.
589     energy\_diff = trial\_energy - old\_energy
590     !write(*,*) "Diferencia de energías: ", energy_diff
591     energy\_change = 0.0
592     accepted = metropolis\_step(energy\_diff/t\_env)
593     if (accepted) then
594         positions(idx, :, :) = trial\_pos
595         chirality(idx) = trial\_chir
596         orientation(idx) = trial\_orientation
597         energy\_change = energy\_diff
598     end if
599
600
601 end subroutine montecarlo
602
603 end program tesis\_canonico
```

E.2. Programa del modelo L en el ensamble gran canónico

```
1  !Este programa solo permite cambiar de quiralidad de manera
2  !horizontal y vertical, SIN rotaciones de 180°
3
4  !Se sustituyó el guion bajo con diagonal-guion bajo para que salgan bien los comentarios
5  program tesis\_canonico
6  implicit none
7
8  !********** DEFINICION DE VARIABLES *********
9  !********** PARAMETROS DEL SISTEMA *********
10 !Estos parametros se pueden dejar en otro archivo de lectura
11 !Los valores de lambda tambien podrian dejarse en otro archivo de lectura
12 integer, parameter :: side = 30, nsample = side**2, cycles = side**5, nsites = side
       **2, ntemps = side
13 real(8), parameter :: l1 = 0.9, l2 = 0.45, init\_temperature = 3.0, final\_temperature
        = 0.3
14 real(8), parameter :: chempot = 40.0 !Potencial quimico
15 real(8) :: lambdawl !Longitud de onda termica
16 real(8) :: zz !Actividad
17 ! En caso de que se quiera modificar o no, pongo el rango de interaccion aparte
18 real(8) :: lambda
19 integer :: nmol = floor(dble(nsites)/10) !Numero de particulas (inicial, luego cambia)
20 integer :: nps, neigh\_number
21 integer :: initial\_iter = 1, final\_iter = 20
22
23 !********** MATRICES A USAR *********
24 real(8), dimension(nsites, 2) :: mol\_centers
25 real(8), dimension(nsites, 3, 2) :: positions
26 integer, dimension(nsites) :: chirality, orientation
27 logical, dimension(nsites) :: occupancy
28 integer, allocatable, dimension(:, :) :: neighbors
```

```
29 real(8), allocatable, dimension(:, :, :) :: neighbor\_pos
30 logical, allocatable, dimension(:) :: neighbor\_occ
31 real(8), dimension(cycles/nsample + 2) :: system\_energy
32
33 !********** VARIABLES INTERNAS *********
34 integer :: ii, jj, kk, sample\_counter, counter, lambda\_iter, size\_seed, cycles\
       _ptemp, temp\_counter
35 real(8) :: energy\_change, energy, size\_sys, eps, zero, pi, temperature, dtemp, quad\
       _a, quad\_b
36 real(8), dimension(ntemps + 1) :: temp\_arr
37 integer :: new\_chirality
38 real(8), dimension(3, 2) :: new\_pos
39 character(70) :: chir\_format, filename, filename2
40 character(40) :: directory
41 real(8), dimension (4, 2) :: cell
42 real(8), dimension(32) :: relative\_dist
43 real(8), dimension(2) :: temp\_center
44 real(8), dimension(20) :: unique\_dist !De aqui se obtienen las lambda
45 integer, dimension(8) :: values
46 integer, dimension(:), allocatable :: seed
47
48
49 !Directorio donde se guardaran los resultados
50 directory = "modif3/"
51 print*, trim(directory)
52
53 !eps es el valor que le sumaba a lambda para evitar errores de redondeo
54 !pero parece que no es necesario
55 eps = 0.0
56
57 !Formato de la matriz dependiente del tamaño
58 !side sustituye a I2
59 !Es para la escritura de la matriz del sistema
60 write(chir\_format, '( "(", I2, "I4)" )') side
61
62 !Calculo de los valores posibles de lambda tal que alcancen todos los sitios
63 !de interacción de las moleculas primeras vecinas
64
65 !Estas son las posible posiciones de los sitios de interacción de una molecula
66 !Los asteriscos son los sitios de interacción
67 ! 2 * * 4
68 ! —
69 ! —
70 ! —
71 !I1 —
72 ! —
73 ! —
74 ! —
75 ! 1 *————* 3
76 ! (0,0) I2
77
78 zero = 0.0 !Es tipo real(8)
```

```
 cell(1,:) = (/zero, zero/)
 cell(2,:) = (/zero, l1/)
 cell(3,:) = (/l2, zero/)
 cell(4,:) = (/l2, l1/)

!Las posiciones de los sitios de interacción de las moleculas mas cercanas
!se encuentran desplazadas en los ejes x, y
!Aqui los asteriscos son los centros de red donde se localizan las moleculas vecinas.
!En cada una esta centrada una molecula como la de arriba.
! * * *
! (-1,1) (1,0) (1,1)


! * * *
! (-1,0) (0,0) (1,0)


! * * *
! (-1,-1) (-1,0) (-1,1)

!Con la información de arriba, se calculan las 8*4 = 32 posibles distancias entre
!sitios de interacción de moleculas vecinas
counter = 1
do jj = -1, 1 !Desplazamiento en el eje y
    do ii = -1, 1 !Desplazamiento en el eje x
        do kk = 1, 4 !Numero de sitios de interacción por molecula
            if (ii == 0 .and. jj == 0) then !La molecula central no se incluye
                cycle
            end if
            temp\_center = (/dble(ii), dble(jj)/)
            relative\_dist(counter) = norm(cell(kk, :) + temp\_center)
            counter = counter + 1
        end do
    end do
end do

!Como hay distancias repetidas, hay que ver cuales son unicas
!Hay 20 distancias unicas
unique\_dist = 0.0
counter = 1
do ii = 1, 32
    if (.not.(any(abs(unique\_dist - relative\_dist(ii)) < 0.00001))) then
! if (.not.(any(unique_dist == relative_dist(ii)))) then
        unique\_dist(counter) = relative\_dist(ii)
        counter = counter + 1
    end if
end do

!Cambiar semilla del PRNG con la fecha y hora del momento de ejecución
!A ver si asi cambia la salida
call date\_and\_time(VALUES=values)
call random\_seed(size=size\_seed)
```

```
131 allocate(seed(size\_seed))
132 seed(:) = values(:)
133 call random\_seed(put=seed)
134
135 !Se realiza una simulación por cada posible distancia entre sitios de interacción
136 !Los valores de lambda son precisamente las posibles distancias
137 do lambda\_iter = initial\_iter, final\_iter
138     lambda = unique\_dist(lambda\_iter) + eps
139     write(*,*) "lambda = ", lambda
140
141     !Numero de vecinos efectivos por "lado"
142     !Es para reducir la busqueda de vecinos
143     !nps: neighbors per side
144     if (lambda <= 1.005) then
145         nps = 1
146     else
147         nps = 2
148     end if
149     !si nps = 1, entonces neigh_number = 8
150 ! * * *
151 ! * 0 *
152 ! * * *
153
154     !si nps = 2, entonces neigh_number = 24
155 ! * * * * *
156 ! * * * * *
157 ! * * 0 * *
158 ! * * * * *
159 ! * * * * *
160
161     neigh\_number = (2*nps + 1)**2 - 1
162
163     !Asignación de memoria despues de calcular neigh_number
164     allocate(neighbors(nsites, neigh\_number))
165     allocate(neighbor\_pos(neigh\_number + 1, 3, 2))
166     allocate(neighbor\_occ(neigh\_number + 1))
167
168     !Inicialización
169     call create\_molecules(nsites, mol\_centers, positions, chirality, orientation,
        occupancy, neighbors)
170
171     !Energia inicial
172     energy = 0.0
173     do jj = 1, nsites
174         ! Posición de la molecula çentral"
175         neighbor\_pos(1, :, :) = positions(jj, :, :)
176         neighbor\_occ(1) = occupancy(jj)
177         ! Se buscan las posiciones de las vecinas
178         do ii = 1, neigh\_number
179             neighbor\_pos(ii + 1, :, :) = positions(neighbors(jj, ii), :, :)
180             neighbor\_occ(ii + 1) = occupancy(neighbors(jj, ii))
181         end do
```

```
182         energy = energy + potential\_energy(neighbor\_pos, neighbor\_occ)
183         !print*, .Energia intermedia :", energy
184     end do
185     energy = energy / 2.0
186     size\_sys = dble(nsites)
187     system\_energy(1) = energy/size\_sys
188     !print*, .Energia inicial: ", system_energy(1)
189
190     ! Ciclo de Montecarlo
191     ! Se modificó para cambiar la temperatura poco a poco
192     dtemp = (init\_temperature - final\_temperature)/(ntemps - 1)
193
194     !Quiero convertir el dt lineal a uno cuadrático, para que la temperatura descienda rapido
195     !al principio y lento al final, a ver si funciona para correr con side**4
196     quad\_a = (init\_temperature - final\_temperature)/(init\_temperature**2 - final\
            _temperature**2)
197     quad\_b = final\_temperature - quad\_a*final\_temperature**2
198     temp\_arr(1) = init\_temperature
199     do ii = 1, (ntemps - 2)
200         temp\_arr(ii + 1) = quad\_a*(init\_temperature - ii*dtemp)**2 + quad\_b
201     end do
202     temp\_arr(ntemps) = final\_temperature
203     temp\_arr(ntemps + 1) = final\_temperature
204
205     temperature = init\_temperature
206     lambdawl = sqrt(1/temperature) !Longitud de onda termica
207     zz = exp(chempot/temperature)/(lambdawl**3) !Actividad
208
209     !Número de ciclos que se hacen por cada temperatura
210     cycles\_ptemp = floor(dble(cycles)/ntemps)
211     sample\_counter = 1
212     temp\_counter = 1
213     do ii = 1, cycles
214         lambdawl = sqrt(1/temperature) !Longitud de onda termica
215         zz = exp(chempot/temperature)/(lambdawl**3) !Actividad
216         call montecarlo(nsites, nmol, mol\_centers, positions, chirality, \&
217             orientation, occupancy, neighbors, temperature, energy\_change)
218
219         energy = energy + energy\_change
220         !Muestreo
221         if (mod(ii, nsample) == 0) then
222             system\_energy(sample\_counter) = energy/size\_sys
223             sample\_counter = sample\_counter + 1
224             !if (ii == cycles/10) then
225             ! write(*,*) .Avance: ", 100*ii/cycles, iend if
226         end if
227         !Bajar la temperatura cada cierto número de ciclos
228         if (mod(ii, cycles\_ptemp) == 0) then
229 ! print*, "Temp. intermedia: ", temperature
230             temp\_counter = temp\_counter + 1
231 ! print*, temp_counter
232             temperature = temp\_arr(temp\_counter)
```

```
233             lambdawl = sqrt(1/temperature) !Longitud de onda termica
234             zz = exp(chempot/temperature)/(lambdawl**3) !Actividad
235         end if
236     end do
237 
238     ! Escritura de archivos
239     ! La siguiente linea es para variar el nombre del archivo conforme a lambda
240     ! Archivo de la configuración del sistema
241     ! Aqui es donde se deberia multiplicar chirality*orientation para
242     ! tener en un solo numero la quiralidad y la orientación
243     write(filename, '( A, "lambda", F7.5, "\_final.txt" )') trim(directory), lambda
244     print*, filename
245     open (10, file = filename, status='UNKNOWN')
246     write(10, chir\_format) chirality*orientation
247     close(10)
248 
249     !Archivo de las energias
250     write(filename2, '( A, "t\_energies\_lambda", F7.5, ".txt" )') trim(directory),
            lambda
251     open(20, file = filename2, status='UNKNOWN')
252     write(20, *) system\_energy
253     close(20)
254 
255     !Deasignación de tamaño, para asignarle otra dependiendo de nps
256     deallocate(neighbors)
257     deallocate(neighbor\_pos)
258     deallocate(neighbor\_occ)
259 end do
260 
261 contains
262 
263 !********** FUNCIONES Y SUBRUTINAS *****************
264 
265 !Norma de un vector
266 !En el argumento iria la resta de dos vectores si es necesario
267 real(8) function norm(vr)
268     implicit none
269 
270     real(8), dimension(2) :: vr
271     norm = sqrt(vr(1)**2 + vr(2)**2)
272 
273 end function norm
274 
275 !Elegir un numero entero entre a y b, incluidos a y b
276 integer function rand\_int(a, b)
277     implicit none
278     integer, intent(in) :: a, b ! a ¡b
279     integer :: diff
280     real(8) :: rand
281 
282     diff = dble(b + 1 - a)
283     call RANDOM\_NUMBER(rand)
```

```
284     rand\_int = floor(diff*rand) + a !El + 1.0 es para que tambien seleccione b
285 
286 end function rand\_int
287 
288 !Pozo cuadrado
289 !El argumento es un vector
290 real(8) function square\_well(r\_ij)
291     implicit none
292 
293     real(8), dimension(2) :: r\_ij
294     real(8) :: ans, dist
295     dist = norm(r\_ij)
296     if (dist == 0.0) then
297         ans = 1000.0 !Es para simular infinitos, tal vez me traiga problemitas
298     else if (dist > 0 .and. dist <= lambda) then
299         ans = -1.0
300     else !dist ¿lambda
301         ans = 0.0
302     end if
303     square\_well = ans
304 end function square\_well
305 
306 !Moleculas derechas
307 !Quiero que la función me regrese una matriz
308 !real(8) function dextro_molecule(square_pos) NO FUNCIONA
309 !real(8), dimension(3, 2) function dextro_molecule(square_pos) NO FUNCIONA
310 function dextro\_molecule(square\_pos, orientation)
311 ! Regresa una lista con las posiciones de los atomos de cada molecula para que sea derecha
312     implicit none
313     real(8), dimension(3, 2) :: pos, dextro\_molecule !Defini el tipo de la función
        ADENTRO, no antes del nombre
314     real(8), dimension(2) :: square\_pos
315     integer :: orientation, ii
316                     ! 3
317     pos(1, 1) = -l2/2.0    ! *
318     pos(1, 2) = -l1/2.0    ! —
319     pos(2, 1) = l2/2.0    ! —
320     pos(2, 2) = -l1/2.0    ! —
321     pos(3, 1) = l2/2.0    ! —
322     pos(3, 2) = l1/2.0    ! *———*
323                     ! 1 2
324 
325     do ii = 1, 3
326         if (orientation == 1) then
327             pos(ii, :) = square\_pos(:) + pos(ii, :)
328         else if (orientation == 2) then
329             ! El signo menos basta para voltearla 180 grados
330             pos(ii, :) = square\_pos(:) - pos(ii, :)
331         else
332             print*, "ERROR DERECHAS"
333         end if
334     end do
```

```
335     !La siguiente linea era lo que me faltaba :v
336     dextro\_molecule = pos
337 end function dextro\_molecule
338 
339 !Moleculas izquierdas
340 function levo\_molecule(square\_pos, orientation)
341 ! Regresa una lista con las posiciones de los atomos de cada molecula para que sea derecha
342     implicit none
343     real(8), dimension(3, 2) :: pos, levo\_molecule !Defini el tipo de la función ADENTRO,
            no antes del nombre
344     real(8), dimension(2) :: square\_pos
345     integer :: orientation, ii
346                     ! 3
347     pos(1, 1) = l2/2.0    ! *
348     pos(1, 2) = -l1/2.0    ! —
349     pos(2, 1) = -l2/2.0    ! —
350     pos(2, 2) = -l1/2.0    ! —
351     pos(3, 1) = -l2/2.0    ! —
352     pos(3, 2) = l1/2.0    ! *————*
353                     ! 2 1
354     do ii = 1, 3
355         if (orientation == 1) then
356             pos(ii, :) = square\_pos(:) + pos(ii, :)
357         else if (orientation == 2) then
358             pos(ii, :) = square\_pos(:) - pos(ii, :)
359         else
360             print*, "ERROR IZQUIERDAS"
361         end if
362     end do
363 
364     levo\_molecule = pos
365 end function levo\_molecule
366 
367 subroutine create\_molecules(nsites, mol\_centers, positions, chirality, orientation,
        occupancy, neighbors)
368     implicit none
369     integer, intent(in) :: nsites
370     real(8), dimension(nsites, 2), intent(out) :: mol\_centers
371     real(8), dimension(nsites, 3, 2), intent(out) :: positions
372     integer, dimension(nsites), intent(out) :: chirality, orientation
373     logical, dimension(nsites), intent(out) :: occupancy
374     integer, dimension(nsites, neigh\_number), intent(out) :: neighbors
375     real(8) :: rand
376     integer :: ii, jj, row, col, mol\_index, counter
377 
378     ! Hacer que "la mitad"de las moleculas tengan la misma quiralidad de manera aleatoria
379     do ii = 1, nsites
380         call RANDOM\_NUMBER(rand)
381         if (rand <= 0.5) then
382             chirality(ii) = 1 !"D"
383         else
384             chirality(ii) = -1 !"L"
```

```
385         end if
386
387         ! Creo que es mejor asi y no con rand_int para decir cual
388         ! orientación es cual
389         call RANDOM\_NUMBER(rand)
390         if (rand < 0.5) then
391             orientation(ii) = 1 !Lado corto abajo
392         else
393             orientation(ii) = 2 !Lado corto arriba
394         end if
395     end do
396
397     !chirality = 1
398
399     ! Situa las moleculas en posiciones y orientaciones aleatorias
400     do jj = 0, (side - 1)
401         do ii = 0, (side - 1)
402             mol\_index = jj*side + ii + 1
403             mol\_centers(mol\_index, 1) = ii
404             mol\_centers(mol\_index, 2) = jj
405             if (chirality(mol\_index) == -1) then
406                 !temp_pos = levo_molecule(mol_center(mol_index, :))
407                 positions(mol\_index, :, :) = \&
408                 levo\_molecule(mol\_centers(mol\_index, :), orientation(mol\_index))
409             else if (chirality(mol\_index) == 1) then
410             !temp_pos = dextro_molecule(mol_center(mol_index, :))
411                 positions(mol\_index, :, :) = \&
412                 dextro\_molecule(mol\_centers(mol\_index, :), orientation(mol\_index))
413             end if
414             !Asignar vecinos
415             !nps esta definido desde arriba
416             counter = 1
417             do col = -nps, nps
418                 do row = -nps, nps
419                     if (col == 0 .and. row == 0) then
420                         cycle  !Se salta la molecula central porque no es su propia vecina
421                     else
422                         neighbors(mol\_index, counter) = modulo(jj + col, side)*side +
                                modulo(ii + row, side) + 1
423                         counter = counter + 1
424                     end if
425                 end do
426             end do
427         end do
428     end do
429
430     !Asignar ocupación de los sitios
431     occupancy = .false.
432     ii = 0
433     do while (ii < nmol) !nmol es el numero (en este momento inicial) de moleculas en la red
434         mol\_index = rand\_int(1, nsites)
435         if (occupancy(mol\_index) .eqv. .false.) then
```

```
436             occupancy(mol\_index) = .true.
437             ii = ii + 1
438         ! print*, "nuevamente ocupado: ", occupancy(mol_index)
439         end if
440         !print*, ———————————————"
441     end do
442 ! print*, "Mols iniciales original: ", nmol
443 ! print*, "Mols iniciales contador: ", ii
444 
445 end subroutine create\_molecules
446 
447 ! Esta función selecciona la quiralidad contraria y elige al azar una orientación,
448 ! que es equivalente a elegir al azar voltear horizontal o verticalmente
449 ! Para 4 posibles rotaciones tal vez deba cambiar el programa
450 subroutine change\_chirality(mol\_chirality, center, new\_chirality, new\_pos, new\
        _orientation)
451 implicit none
452 integer, intent(in) :: mol\_chirality
453 real(8), dimension(2), intent(in) :: center
454 integer, intent(out) :: new\_chirality, new\_orientation
455 real(8), dimension(3, 2), intent(out) :: new\_pos
456 real(8) :: rand
457 
458     ! Se escoge al azar si se voltea sobre
459     ! el lado largo o el corto
460     new\_orientation = rand\_int(1, 2)
461 
462     !Si es derecha, cambiar a izquierda
463     if (mol\_chirality == -1) then
464         new\_pos = dextro\_molecule(center, new\_orientation)
465     else if (mol\_chirality == 1) then
466         new\_pos = levo\_molecule(center, new\_orientation)
467     end if
468     new\_chirality = -mol\_chirality
469 
470 end subroutine change\_chirality
471 
472 !Esta función rota 180 grados una particula, sin cambiar su quiralidad
473 subroutine rotate\_molecule(mol\_chirality, center, old\_orientation, new\_pos, new\
        _orientation)
474     integer, intent(in) :: mol\_chirality, old\_orientation
475     real(8), dimension(2), intent(in) :: center
476     real(8), dimension(3, 2), intent(out) :: new\_pos
477     integer, intent(out) :: new\_orientation
478 
479     !Conserva quiralidad, solo rota
480     if (mol\_chirality == 1) then
481         if (old\_orientation == 1) then
482             new\_orientation = 2
483             new\_pos = dextro\_molecule(center, new\_orientation)
484         else if (old\_orientation == 2) then
485             new\_orientation = 1
```

```
            new\_pos = dextro\_molecule(center, new\_orientation)
        end if
    else if (mol\_chirality == -1) then
        if (old\_orientation == 1) then
            new\_orientation = 2
            new\_pos = levo\_molecule(center, new\_orientation)
        else if (old\_orientation == 2) then
            new\_orientation = 1
            new\_pos = levo\_molecule(center, new\_orientation)
        end if
    end if
end subroutine rotate\_molecule

subroutine add\_molecule(occupancy, mol\_centers, mol\_index, new\_pos, new\_chirality
    , new\_orientation)
logical, dimension(nsites), intent(in) :: occupancy
real(8), dimension(nsites, 2), intent(in) :: mol\_centers
integer, intent(out) :: mol\_index, new\_chirality, new\_orientation
real(8), dimension(3, 2), intent(out) :: new\_pos
integer, dimension(nsites) :: free\_sites
integer :: ii, counter

real(8) :: rand

!Encontrar indices de sitios libres
counter = 0
free\_sites = 0
do ii = 1, nsites
    if (occupancy(ii) .eqv. .false.) then
        counter = counter + 1
        free\_sites(counter) = ii
    end if
end do

!print*, "Sitios libres"
!print*, free_sites
!print*, Ïndice: ", mol_index

mol\_index = free\_sites(rand\_int(1, counter))
new\_orientation = rand\_int(1, 2) !Orientacion al azar
call RANDOM\_NUMBER(rand)
if (rand < 0.5) then
    new\_pos = dextro\_molecule(mol\_centers(mol\_index, :), new\_orientation)
    new\_chirality = 1
else
    new\_pos = levo\_molecule(mol\_centers(mol\_index, :), new\_orientation)
    new\_chirality = -1
end if
end subroutine add\_molecule

subroutine delete\_molecule(occupancy, mol\_index)
logical, dimension(nsites), intent(in) :: occupancy
```

```
537 integer, intent(out) :: mol\_index
538 integer, dimension(nsites) :: occupied\_sites
539 integer :: ii, counter
540
541 !Encontrar indices de sitios ocupados
542 counter = 0
543 occupied\_sites = 0
544 do ii = 1, nsites
545     if (occupancy(ii)) then
546         counter = counter + 1
547         occupied\_sites(counter) = ii
548     end if
549 end do
550
551 mol\_index = occupied\_sites(rand\_int(1, counter))
552 end subroutine delete\_molecule
553
554 subroutine move\_molecule(occupancy, old\_index, new\_index)
555 logical, dimension(nsites), intent(in) :: occupancy
556 integer, intent(out) :: old\_index, new\_index
557 integer, dimension(nsites) :: free\_sites, occupied\_sites
558 integer :: ii, occupied\_counter, free\_counter
559
560 !Encontrar indices de sitios ocupados y libres
561
562 occupied\_sites = 0
563 free\_sites = 0
564 occupied\_counter = 0
565 free\_counter = 0
566 do ii = 1, nsites
567     if (occupancy(ii)) then
568         occupied\_counter = occupied\_counter + 1
569         occupied\_sites(occupied\_counter) = ii
570     else
571         free\_counter = free\_counter + 1
572         free\_sites(free\_counter) = ii
573     end if
574 end do
575
576 old\_index = occupied\_sites(rand\_int(1, occupied\_counter))
577 new\_index = free\_sites(rand\_int(1, free\_counter))
578 end subroutine move\_molecule
579
580 real(8) function potential\_energy(positions, occupancies)
581     implicit none
582     ! positions incluye las posiciones de cada atomo
583     ! positions(1) es la molecula central
584
585     real(8), dimension(neigh\_number + 1, 3, 2) :: positions
586     logical, dimension(neigh\_number + 1) :: occupancies
587     integer :: ii, jj, kk, aa
588     real(8) :: energy, temp\_dist
```

```
589     real(8), dimension(2) :: dist
590 
591     energy = 0.0
592     do kk = 2, (neigh\_number + 1) !Iterar sobre las particulas vecinas
593         ! Si el sitio central esta vacio, se sale del ciclo y energy = 0
594             if (occupancies(1) .eqv. .false.) then
595             exit
596         end if
597         !print*, "Vecino relativo: ", kk
598         if (occupancies(kk)) then !Se hace el siguiente ciclo solo si hay vecina
599             do jj = 1, 3 ! Sobre los sitios atractivos de las vecinas
600                 do ii = 1, 3 !Sobre las de la particula central
601                     ! Sin condiciones periódicas
602                     !energy += square_well(positions[1][ii] - positions[kk][jj])
603 
604                         !Con condiciones periódicas
605                     do aa = 1, 2 !Coordenadas de los sitios
606                         temp\_dist = abs(positions(1, ii, aa) - positions(kk, jj, aa))
607                         if (temp\_dist > side/2) then
608                             temp\_dist = side - temp\_dist
609                         end if
610                         dist(aa) = temp\_dist
611                     end do
612                     !print*, "Distancia: ", dist
613                     energy = energy + square\_well(dist)
614                 end do
615             end do
616         end if
617         !print*, "Energia:", energy
618     end do
619     potential\_energy = energy
620 end function potential\_energy
621 
622 !Criterio de Metropolis
623 logical function metropolis\_step(exp\_arg)
624 !Returns True if the trial configuration goes towards a region with higher probability
625 !or it is given the chance to explore regions with fewer probability
626 
627 implicit none
628 
629 real(8) :: exp\_arg, w, rand
630 logical :: accepted
631 
632     if (exp\_arg < 0.0) then
633         accepted = .true. ! Se actualiza el estado del sistema
634     else
635         w = exp(-exp\_arg)
636         call RANDOM\_NUMBER(rand)
637         if (rand < w) then ! Energia: w
638             accepted = .true. ! Tambien se actualiza el estado del sistema
639     else
640         accepted = .false.
```

```
641         end if
642     end if
643     metropolis\_step = accepted
644
645 end function metropolis\_step
646
647 subroutine montecarlo(nsites, nmol, centers, positions, chirality, orientation, \&
648             occupancy, neighbors, t\_env, energy\_change)
649     implicit none
650     ! Variables de entrada y salida
651     integer, intent(in) :: nsites
652     integer, intent(inout) :: nmol
653     real(8), dimension(nsites, 2), intent(in) :: centers
654     integer, dimension(nsites, neigh\_number), intent(in) :: neighbors
655     real(8), dimension(nsites, 3, 2), intent(inout) :: positions
656     integer, dimension(nsites), intent(inout) :: chirality, orientation
657     logical, dimension(nsites), intent(inout) :: occupancy
658     real(8), intent(in) :: t\_env
659     real(8), intent(out) :: energy\_change
660
661     !Variables internas
662     integer :: ii, old\_index, new\_index, neighbor, trial\_chir
663     real(8), dimension(neigh\_number + 1, 3, 2) :: neighbor\_pos, neighbor\_pos2
664     logical, dimension(neigh\_number + 1) :: neighbor\_occ, neighbor\_occ2
665     real(8), dimension(3, 2) :: old\_pos, new\_pos
666     real(8) :: old\_energy, trial\_energy, energy\_diff, rand
667     integer :: new\_orientation, new\_chirality
668     logical :: accepted
669     real(8) :: p\_add, p\_del, p\_move, p\_rot, p\_change\_chir
670     real(8), dimension(5) :: pv
671
672     p\_add = 0.1 ! Probabilidad de agregar una molecula
673     p\_del = 0.1 ! Prob. de remover
674     p\_move = 0.1 ! Prob. de desplazar
675     p\_rot = 0.35 ! Prob. de rotar
676     p\_change\_chir = 0.35 ! Prob. de cambiar de quiralidad
677
678     !Las probs. anteriores tienen que sumar 1 (la prob. de intentar un cambio en el sistema)
679     pv(1) = p\_add
680     pv(2) = pv(1) + p\_del
681     pv(3) = pv(2) + p\_move
682     pv(4) = pv(3) + p\_rot
683     pv(5) = pv(4) + p\_change\_chir
684
685     if (pv(5) < 0.00001) then
686         print*, "Las probabilidades del metodo de Montecarlo no suman 1"
687     end if
688
689     ! Intento de cambio: elegir entre agregar, remover, desplazar, rotar o cambiar la quiralidad de una
            molecula
690     call RANDOM\_NUMBER(rand)
691
```

```
692     !Intentar agregar una molecula en un sitio vacio
693     energy\_change = 0.0
694     if (rand <= pv(1)) then
695         !Si hay sitios vacios, se puede agregar una molecula
696         if (nmol < nsites) then
697             !print*, "Agregar"
698             call add\_molecule(occupancy, centers, new\_index, \&
699                 new\_pos, new\_chirality, new\_orientation)
700             neighbor\_pos2(1, :, :) = new\_pos
701             neighbor\_occ2(1) = .true.
702             !write(*,*) "Posición: ", positions(idx, :, :)
703             do ii = 1, neigh\_number
704                 neighbor = neighbors(new\_index, ii)
705                 neighbor\_pos2(ii + 1, :, :) = positions(neighbor, :, :)
706                 neighbor\_occ2(ii + 1) = occupancy(neighbor)
707             end do
708
709             old\_energy = 0.0 !No hay interacciones proque el sitio esta vacio
710             trial\_energy = potential\_energy(neighbor\_pos2, neighbor\_occ2)
711             energy\_diff = trial\_energy - old\_energy
712             accepted = metropolis\_step(energy\_diff/t\_env - \&
713                       log(zz*side*side/(nmol + 1)))
714             if (accepted) then
715                 positions(new\_index, :, :) = new\_pos
716                 chirality(new\_index) = new\_chirality
717                 orientation(new\_index) = new\_orientation
718                 occupancy(new\_index) = .true.
719                 energy\_change = energy\_diff
720                 nmol = nmol + 1
721             !else
722             ! energy_change = 0.0
723             end if
724         end if
725
726     !Intentar eliminar una molecula del sistema
727     else if ((pv(1) < rand) .and. (rand <= pv(2))) then
728         if (nmol > 0) then
729             !print*, "Eliminar"
730             call delete\_molecule(occupancy, old\_index)
731             neighbor\_pos(1, :, :) = positions(old\_index, :, :)
732             neighbor\_occ(1) = .true.
733             do ii = 1, neigh\_number
734                 neighbor = neighbors(old\_index, ii)
735                 neighbor\_pos(ii + 1, :, :) = positions(neighbor, :, :)
736                 neighbor\_occ(ii + 1) = occupancy(neighbor)
737             end do
738             old\_energy = potential\_energy(neighbor\_pos, neighbor\_occ)
739             trial\_energy = 0.0
740             energy\_diff = trial\_energy - old\_energy
741             accepted = metropolis\_step(energy\_diff/t\_env - \&
742                       log(nmol/(zz*side*side)))
743             if (accepted) then
```

```
744                 occupancy(old\_index) = .false.
745                 energy\_change = energy\_diff
746                 nmol = nmol - 1
747             !else
748             ! energy_change = 0.0
749             end if
750         end if
751
752     ! Intentar desplazar una particula existente a un sitio vacio
753     else if ((pv(2) < rand) .and. (rand <= pv(3))) then
754         !Si hay moleculas y sitios vacios, intentar mover una molecula
755         !En caso contrario no se podra hacer nada
756         if ((0 < nmol) .and. (nmol < nsites)) then
757             !print*, "Desplazar"
758             call move\_molecule(occupancy, old\_index, new\_index)
759             !print*, "Índice actual: ", old_index
760             !print*, "Índice de prueba: ", new_index
761
762
763 ! Parece que funciona excepto si es vecina de la molecula de prueba
764             neighbor\_pos(1, :, :) = positions(old\_index, :, :)
765             neighbor\_occ(1) = .true.
766             do ii = 1, neigh\_number
767                 neighbor = neighbors(old\_index, ii)
768                 neighbor\_pos(ii + 1, :, :) = positions(neighbor, :, :)
769                 neighbor\_occ(ii + 1) = occupancy(neighbor)
770             end do
771
772 ! No se si es el error que busco pero el hecho es que no estoy poniendo la posicion inicial en el sitio de
        prueba, solo estoy como "desapareciendo"la original e "insertando"la de prueba en otro lado, pero
        sin la config. de la inicial.
773             !Este es el error que estaba comentiendo:
774             !neighbor_pos2(1, :, :) = positions(new_index, :, :)
775             !La particula desplazada debe tener la quiralidad y orientacion
776             ! de la original pero ubicada en el nuevo sitio
777             if (chirality(old\_index) == 1) then
778                 neighbor\_pos2(1, :, :) = \&
779                 dextro\_molecule(centers(new\_index, :), orientation(old\_index))
780             else if (chirality(old\_index) == -1) then
781                 neighbor\_pos2(1, :, :) = \&
782                 levo\_molecule(centers(new\_index, :), orientation(old\_index))
783             end if
784
785             neighbor\_occ2(1) = .true.
786             do ii = 1, neigh\_number
787                 neighbor = neighbors(new\_index, ii)
788                 neighbor\_pos2(ii + 1, :, :) = positions(neighbor, :, :)
789                 neighbor\_occ2(ii + 1) = occupancy(neighbor)
790                 if (neighbor == old\_index) then
791                     neighbor\_occ2(ii + 1) = .false.
792                 end if
793             end do
```

```
794
795             old\_energy = potential\_energy(neighbor\_pos, neighbor\_occ)
796             !print*, .Energia actual: ", old_energy
797             trial\_energy = potential\_energy(neighbor\_pos2, neighbor\_occ2)
798             !print*, .Energia de prueba: ", trial_energy
799             energy\_diff = trial\_energy - old\_energy
800             accepted = metropolis\_step((trial\_energy - old\_energy)/t\_env)
801             if (accepted) then
802                 !ERROR:
803                 !positions(new_index, :, :) = positions(old_index, :, :)
804                 positions(new\_index, :, :) = neighbor\_pos2(1, :, :)
805                 chirality(new\_index) = chirality(old\_index)
806                 occupancy(old\_index) = .false.
807                 occupancy(new\_index) = .true.
808                 energy\_change = energy\_diff
809             !else
810             ! energy_change = 0.0
811             end if
812         end if
813
814     ! Intentar rotar
815     else if ((pv(3) < rand) .and. (rand <= pv(4))) then
816         if (nmol > 0) then
817             !print*, Rotar"
818             ! delete_molecule solo regresa un indice de un sitio ocupado,
819             ! que es lo que se necesita en este momento
820             call delete\_molecule(occupancy, old\_index)
821             call rotate\_molecule(chirality(old\_index), centers(old\_index, :),
                    orientation(old\_index), new\_pos, new\_orientation)
822             neighbor\_pos(1, :, :) = positions(old\_index, :, :)
823             neighbor\_occ(1) = .true.
824             !write(*,*) "Posición: ", positions(idx, :, :)
825             do ii = 1, neigh\_number
826                 neighbor = neighbors(old\_index, ii)
827                 neighbor\_pos(ii + 1, :, :) = positions(neighbor, :, :)
828             end do
829
830             neighbor\_pos2(1, :, :) = new\_pos
831             neighbor\_pos2(2:(neigh\_number + 1), :, :) = \&
832                       neighbor\_pos2(2:(neigh\_number + 1), :, :)
833             neighbor\_occ2 = neighbor\_occ
834
835             old\_energy = potential\_energy(neighbor\_pos, neighbor\_occ)
836             trial\_energy = potential\_energy(neighbor\_pos2, neighbor\_occ2)
837             energy\_diff = trial\_energy - old\_energy
838             accepted = metropolis\_step((trial\_energy - old\_energy)/t\_env)
839             if (accepted) then
840                 positions(old\_index, :, :) = new\_pos
841                 orientation(old\_index) = new\_orientation
842                 energy\_change = energy\_diff
843             end if
844         end if
```

```
845
846     !Intentar cambiar la quiralidad
847     else if (pv(4) < rand) then
848         if (nmol > 0) then
849             !print*, "Quiralidad"
850             ! delete_molecule solo regresa un indice de un sitio ocupado,
851             ! que es lo que se necesita en este momento
852             call delete\_molecule(occupancy, old\_index)
853             call change\_chirality(chirality(old\_index), \&
854             centers(old\_index, :), new\_chirality, new\_pos, new\_orientation)
855             neighbor\_pos(1, :, :) = positions(old\_index, :, :)
856             neighbor\_occ(1) = .true.
857             !write(*,*) "Posición: ", positions(idx, :, :)
858             do ii = 1, neigh\_number
859                 neighbor = neighbors(old\_index, ii)
860                 neighbor\_pos(ii + 1, :, :) = positions(neighbor, :, :)
861             end do
862
863             neighbor\_pos2(1, :, :) = new\_pos
864             neighbor\_pos2(2:(neigh\_number + 1), :, :) = \&
865                         neighbor\_pos(2:(neigh\_number + 1), :, :)
866             neighbor\_occ2 = neighbor\_occ
867
868             old\_energy = potential\_energy(neighbor\_pos, neighbor\_occ)
869             trial\_energy = potential\_energy(neighbor\_pos2, neighbor\_occ2)
870             energy\_diff = trial\_energy - old\_energy
871
872             accepted = metropolis\_step(energy\_diff/t\_env)
873             if (accepted) then
874                 positions(old\_index, :, :) = new\_pos
875                 chirality(old\_index) = new\_chirality
876                 orientation(old\_index) = new\_orientation
877                 energy\_change = energy\_diff
878             !else
879             ! energy_change = 0.0
880             end if
881         end if
882     end if
883
884 end subroutine montecarlo
885
886 end program tesis\_canonico
```

E.3. Código para crear algunas configuraciones y calcular sus energías

```
1  !Este programa solo permite cambiar de quiralidad de manera
2  !horizontal y vertical, SIN rotaciones de 180°
3  program tesis\_canonico
4  implicit none
5
```

```
!********** DEFINICION DE VARIABLES *********
!********** PARAMETROS DEL SISTEMA *********
!Estos parametros se pueden dejar en otro archivo de lectura
!Los valores de lambda tambien podrian dejarse en otro archivo de lectura
integer, parameter :: side = 50, nsample = side**2, cycles = side**4, nmol = side**2
real(8), parameter :: l1 = 0.9, l2 = 0.45, temperature = 0.1
! En caso de que se quiera modificar o no, pongo el rango de interaccion aparte
real(8) :: lambda

integer :: nps, neigh\_number

!********** MATRICES A USAR *********
real(8), dimension(nmol, 2) :: mol\_centers
real(8), dimension(nmol, 3, 2) :: positions
integer, dimension(nmol) :: chirality, orientation
integer, allocatable, dimension(:, :) :: neighbors
real(8), allocatable, dimension(:, :, :) :: neighbor\_pos
real(8), dimension(cycles/nsample + 2) :: system\_energy

!********** VARIABLES INTERNAS *********
integer :: ii, jj, kk, sample\_counter, counter, lambda\_iter, size\_seed
real(8) :: energy\_change, energy, size\_sys, eps, zero, pi
integer :: new\_chirality
real(8), dimension(3, 2) :: new\_pos
character(70) :: chir\_format, filename, filename2
character(40) :: directory
real(8), dimension (4, 2) :: cell
real(8), dimension(32) :: relative\_dist
real(8), dimension(2) :: temp\_center
real(8), dimension(20) :: unique\_dist !De aqui se obtienen las lambda
integer, dimension(8) :: values
integer, dimension(:), allocatable :: seed


!Directorio donde se guardaran los resultados
!El numero de caracteres esta contado: 17
directory = "quiralidad\_rotacion\_eps/"
print*, directory

!eps es el valor que le sumaba a lambda para evitar errores de redondeo
!pero parece que no es necesario
eps = 0.0

!Formato de la matriz dependiente del tamaño
!side sustituye a I2
!Es para la escritura de la matriz del sistema
write(chir\_format, '( "(", I2, "I4)" )') side

!Calculo de los valores posibles de lambda tal que alcancen todos los sitios
!de interacción de las moleculas primeras vecinas

!Estas son las posible posiciones de los sitios de interacción de una molecula
```

```
!Los asteriscos son los sitios de interacción
! 2 * * 4
! —
! —
! —
! l1 —
! —
! —
! —
! 1 *———* 3
! (0,0) l2

zero = 0.0 !Es tipo real(8)
 cell(1,:) = (/zero, zero/)
 cell(2,:) = (/zero, l1/)
 cell(3,:) = (/l2, zero/)
 cell(4,:) = (/l2, l1/)

!Las posiciones de los sitios de interacción de las moleculas mas cercanas
!se encuentran desplazadas en los ejes x, y
!Aqui los asteriscos son los centros de red donde se localizan las moleculas vecinas.
!En cada una esta centrada una molecula como la de arriba.
! * * *
! (-1,1) (1,0) (1,1)


! * * *
! (-1,0) (0,0) (1,0)


! * * *
! (-1,-1) (-1,0) (-1,1)

!Con la información de arriba, se calculan las 8*4 = 32 posibles distancias entre
!sitios de interacción de moleculas vecinas
counter = 1
do jj = -1, 1 !Desplazamiento en el eje y
    do ii = -1, 1 !Desplazamiento en el eje x
        do kk = 1, 4 !Numero de sitios de interacción por molecula
            if (ii == 0 .and. jj == 0) then !La molecula central no se incluye
                cycle
            end if
            temp\_center = (/dble(ii), dble(jj)/)
            relative\_dist(counter) = norm(cell(kk, :) + temp\_center)
            counter = counter + 1
        end do
    end do
end do

!write(*,*) "Distancias"
!write(*, "(F20.18)") relative_dist
```

```
109
110 !Como hay distancias repetidas, hay que ver cuales son unicas
111 !Hay 20 distancias unicas
112 unique\_dist = 0.0
113 counter = 1
114 do ii = 1, 32
115     if (.not.(any(abs(unique\_dist - relative\_dist(ii)) < 0.00001))) then
116 ! if (.not.(any(unique_dist == relative_dist(ii)))) then
117         unique\_dist(counter) = relative\_dist(ii)
118         counter = counter + 1
119     end if
120 end do
121
122 !Cambiar semilla del PRNG con la fecha y hora del momento de ejecución
123 !A ver si asi cambia la salida
124 call date\_and\_time(VALUES=values)
125 call random\_seed(size=size\_seed)
126 allocate(seed(size\_seed))
127 seed(:) = values(:)
128 call random\_seed(put=seed)
129
130
131 !write(*,*) "Distancias unicas"
132 !write(*, "(F20.18)") unique_dist
133
134 !Se realiza una simulación por cada posible distancia entre sitios de interacción
135 !Los valores de lambda son precisamente las posibles distancias
136 do lambda\_iter = 1, 20
137
138     lambda = unique\_dist(lambda\_iter) + eps
139
140     !Numero de vecinos efectivos por "lado"
141     !Es para reducir la busqueda de vecinos
142     !nps: neighbors per side
143     if (lambda <= 1.005) then
144         nps = 1
145     else
146         nps = 2
147     end if
148     !si nps = 1, entonces neigh_number = 8
149 ! * * *
150 ! * 0 *
151 ! * * *
152
153     !si nps = 2, entonces neigh_number = 24
154 ! * * * * *
155 ! * * * * *
156 ! * * 0 * *
157 ! * * * * *
158 ! * * * * *
159
```

```
160     neigh\_number = (2*nps + 1)**2 - 1
161
162     !Asignación de memoria despues de calcular neigh_number
163     allocate(neighbors(nmol, neigh\_number))
164     allocate(neighbor\_pos(neigh\_number + 1, 3, 2))
165
166     !Inicialización
167     call create\_molecules(nmol, mol\_centers, positions, chirality, orientation,
            neighbors)
168     if (lambda\_iter == 1) then
169         write(*, chir\_format) chirality*orientation
170     end if
171
172     !Energia inicial
173     energy = 0.0
174     do jj = 1, nmol
175         ! Posición de la molecula çentral"
176         neighbor\_pos(1, :, :) = positions(jj, :, :)
177         ! Se buscan las posiciones de las vecinas
178         do ii = 1, neigh\_number
179             neighbor\_pos(ii + 1, :, :) = positions(neighbors(jj, ii), :, :)
180         end do
181         energy = energy + potential\_energy(neighbor\_pos)
182     end do
183     energy = energy / 2.0
184     size\_sys = dble(side*side)
185     print*, "Energia de ", lambda, " :", energy/size\_sys
186
187     !Deasignación de tamaño, para asignarle otra dependiendo de nps
188     deallocate(neighbors)
189     deallocate(neighbor\_pos)
190 end do
191 !write(*,*) "Energia inicial: ", system_energy(1)
192 !write(*,*) "Energia final: ", system_energy(nsample**3)
193
194 contains
195
196 !********** FUNCIONES Y SUBRUTINAS *****************
197
198 !Norma de un vector
199 !En el argumento iria la resta de dos vectores si es necesario
200 real(8) function norm(vr)
201     implicit none
202
203     real(8), dimension(2) :: vr
204     norm = sqrt(vr(1)**2 + vr(2)**2)
205
206 end function norm
207
208 !Elegir un numero entero entre a y b, incluidos a y b
209 integer function rand\_int(a, b)
210     implicit none
```

```
211     integer, intent(in) :: a, b ! a ¡b
212     integer :: diff
213     real(8) :: rand
214 
215     diff = dble(b + 1 - a)
216     call RANDOM\_NUMBER(rand)
217     rand\_int = floor(diff*rand) + a !El + 1.0 es para que tambien seleccione b
218 
219 end function rand\_int
220 
221 !Pozo cuadrado
222 !El argumento es un vector
223 real(8) function square\_well(r\_ij)
224     implicit none
225 
226     real(8), dimension(2) :: r\_ij
227     real(8) :: ans, dist
228     dist = norm(r\_ij)
229     if (dist == 0.0) then
230         ans = 1000.0 !Es para simular infinitos, tal vez me traiga problemitas
231     else if (dist > 0 .and. dist <= lambda) then
232         ans = -1.0
233     else !dist ¿lambda
234         ans = 0.0
235     end if
236     square\_well = ans
237 end function square\_well
238 
239 
240 !Moleculas derechas
241 !Quiero que la función me regrese una matriz
242 !real(8) function dextro_molecule(square_pos) NO FUNCIONA
243 !real(8), dimension(3, 2) function dextro_molecule(square_pos) NO FUNCIONA
244 function dextro\_molecule(square\_pos, orientation)
245 ! Regresa una lista con las posiciones de los atomos de cada molecula para que sea derecha
246     implicit none
247     real(8), dimension(3, 2) :: pos, dextro\_molecule !Defini el tipo de la función
        ADENTRO, no antes del nombre
248     real(8), dimension(2) :: square\_pos
249     integer :: orientation, ii
250                     ! 3
251     pos(1, 1) = -l2/2.0    ! *
252     pos(1, 2) = -l1/2.0    ! —
253     pos(2, 1) = l2/2.0    ! —
254     pos(2, 2) = -l1/2.0    ! —
255     pos(3, 1) = l2/2.0    ! —
256     pos(3, 2) = l1/2.0    ! *———*
257                     ! 1 2
258 
259     do ii = 1, 3
260         if (orientation == 1) then
261             pos(ii, :) = square\_pos(:) + pos(ii, :)
```

```
262         else if (orientation == 2) then
263             ! El signo menos basta para voltearla 180 grados
264             pos(ii, :) = square\_pos(:) - pos(ii, :)
265         else
266             print*, "ERROR DERECHAS"
267         end if
268     end do
269     !La siguiente linea era lo que me faltaba :v
270     dextro\_molecule = pos
271 end function dextro\_molecule
272 
273 !Moleculas izquierdas
274 function levo\_molecule(square\_pos, orientation)
275 ! Regresa una lista con las posiciones de los atomos de cada molecula para que sea derecha
276     implicit none
277     real(8), dimension(3, 2) :: pos, levo\_molecule !Defini el tipo de la función ADENTRO,
            no antes del nombre
278     real(8), dimension(2) :: square\_pos
279     integer :: orientation, ii
280                     ! 3
281     pos(1, 1) = l2/2.0    ! *
282     pos(1, 2) = -l1/2.0    ! —
283     pos(2, 1) = -l2/2.0    ! —
284     pos(2, 2) = -l1/2.0    ! —
285     pos(3, 1) = -l2/2.0    ! —
286     pos(3, 2) = l1/2.0    ! *____*
287                     ! 2 1
288     do ii = 1, 3
289         if (orientation == 1) then
290             pos(ii, :) = square\_pos(:) + pos(ii, :)
291         else if (orientation == 2) then
292             pos(ii, :) = square\_pos(:) - pos(ii, :)
293         else
294             print*, "ERROR IZQUIERDAS"
295         end if
296     end do
297 
298     levo\_molecule = pos
299 end function levo\_molecule
300 
301 subroutine create\_molecules(nmol, mol\_centers, positions, chirality, orientation,
        neighbors)
302     implicit none
303     integer, intent(in) :: nmol
304     real(8), dimension(nmol, 2), intent(out) :: mol\_centers
305     real(8), dimension(nmol, 3, 2), intent(out) :: positions
306     integer, dimension(nmol), intent(out) :: chirality, orientation
307     integer, dimension(nmol, neigh\_number), intent(out) :: neighbors
308     real(8) :: rand
309     integer :: ii, jj, row, col, mol\_index, counter
310 
311     ! Hacer que "la mitad"de las moleculas tengan la misma quiralidad de manera aleatoria
```

```
312     do ii = 1, nmol
313         !call RANDOM_NUMBER(rand)
314         !if (rand ¡0.5) then
315             chirality(ii) = 1 !"D"
316         !else
317         ! chirality(ii) = -1 !"L"
318         !end if
319 
320         ! Creo que es mejor asi y no con rand_int para decir cual
321         ! orientación es cual
322         !call RANDOM_NUMBER(rand)
323         !Generar bandas verticales
324         if (mod(floor(dble(ii - 1)/dble(side)), 2) == 0) then
325             if (mod(ii, 2) == 0) then
326             orientation(ii) = 1 !Lado corto abajo
327         else
328             orientation(ii) = 2 !Lado corto arriba
329         end if
330 
331         else
332             if (mod(ii, 2) == 0) then
333             orientation(ii) = 2 !Lado corto abajo
334         else
335             orientation(ii) = 1 !Lado corto arriba
336             end if
337         end if
338 
339     end do
340 
341     !chirality = 1
342 
343     ! Situa las moleculas en posiciones y orientaciones aleatorias
344     do jj = 0, (side - 1)
345         do ii = 0, (side - 1)
346             mol\_index = jj*side + ii + 1
347             mol\_centers(mol\_index, 1) = ii
348             mol\_centers(mol\_index, 2) = jj
349             if (chirality(mol\_index) == -1) then
350                 !temp_pos = levo_molecule(mol_center(mol_index, :))
351                 positions(mol\_index, :, :) = \&
352                 levo\_molecule(mol\_centers(mol\_index, :), orientation(mol\_index))
353             else if (chirality(mol\_index) == 1) then
354             !temp_pos = dextro_molecule(mol_center(mol_index, :))
355                 positions(mol\_index, :, :) = \&
356                 dextro\_molecule(mol\_centers(mol\_index, :), orientation(mol\_index))
357             end if
358             !Asignar vecinos
359             !nps esta definido desde arriba
360             counter = 1
361             do col = -nps, nps
362                 do row = -nps, nps
363                     if (col == 0 .and. row == 0) then
```

```
364                     cycle  !Se salta la molecula central porque no es su propia vecina
365                 else
366                     neighbors(mol\_index, counter) = modulo(jj + col, side)*side +
                            modulo(ii + row, side) + 1
367                     counter = counter + 1
368                 end if
369             end do
370         end do
371     end do
372 end do
373 end subroutine create\_molecules
374
375
376 real(8) function potential\_energy(positions)
377     implicit none
378     ! positions incluye las posiciones de cada atomo
379     ! positions[1] es la molecula central
380
381     real(8), dimension(neigh\_number + 1, 3, 2) :: positions
382     integer :: ii, jj, kk, aa
383     real(8) :: energy, temp\_dist
384     real(8), dimension(2) :: dist
385
386     energy = 0.0
387     do kk = 2, (neigh\_number + 1) !Iterar sobre las particulas vecinas
388         do jj = 1, 3 ! Sobre los sitios atractivos de las vecinas
389             do ii = 1, 3 !Sobre las de la particula central
390                 ! Sin condiciones periódicas
391                 !energy += square_well(positions[1][ii] - positions[kk][jj])
392
393                     !Con condiciones periódicas
394                 do aa = 1, 2 !Coordenadas de los sitios
395                     temp\_dist = abs(positions(1, ii, aa) - positions(kk, jj, aa))
396                     if (temp\_dist > side/2) then
397                         temp\_dist = side - temp\_dist
398                     end if
399                     dist(aa) = temp\_dist
400                 end do
401                 energy = energy + square\_well(dist)
402             end do
403         end do
404     end do
405     potential\_energy = energy
406 end function potential\_energy
407
408 end program tesis\_canonico
```

E.4. Código para calcular exceso enantiomérico y tamaño y número de cúmulos

```
1 function count_cluster(system)
```

```
    nmols = length(system)
    is_bond = fill(false, nmols)
    is_checked = fill(false, nmols)
    chirality = system
    neighbors = Vector(nmols)
    cluster = zeros(Int64, nmols)


    side = floor(Int64, sqrt(nmols))
    for jj in 0:(side - 1)
        for ii in 0:(side - 1)
            mol_index = jj*side + ii + 1

            upper = mod(jj + 1, side)*side + ii + 1
            lower = mod(jj - 1, side)*side + ii + 1
            left = jj*side + mod(ii - 1, side) + 1
            right = jj*side + mod(ii + 1, side) + 1
            neighbors[mol_index] = [upper, lower, left, right]
        end
    end

    #Crear varios cúmulos
    ncluster = 1
    while sum(is_checked) < nmols
        idx = 0
        for ii in 1:nmols
            if cluster[ii] == 0 && !is_checked[ii] #if !is_checked
                idx = ii
                cluster[idx] = ncluster
                break
            end
        end
# Crear un cúmulo
    # Si todos los sitios ünidosza han sido revisados, entonces ya se formó un cúmulo
    # Siempre habrá menor o igual número de sitios revisados que de unidos, por lo que si
    # no se cumple la condición, ya se revisaron todos los sitios
        while is_checked[idx] == false
            for nb in neighbors[idx]
                # Intentar unir dos sitios vecinos
                if (is_checked[nb] == false && chirality[nb] == chirality[idx])
                     is_bond[idx] = true
                     is_bond[nb] = true
                     cluster[nb] = ncluster
                end
            end
            is_checked[idx] = true
        # Escoger uno de los sitios del cúmulo para revisar sus vecinos
            for ii in 1:nmols
                if cluster[ii] == ncluster && !is_checked[ii]
                    idx = ii
                    break
                end
            end
```

```
55          end
56
57          ncluster = ncluster + 1
58      end
59      return cluster
60  end
61
62  direc_arriba = pwd()*"/lado50_tiempo5_enfr/"
63  directs = readdir(direc_arriba);
64
65  for direct in directs
66  println("Directorio actual: "*direct)
67  direc = direc_arriba*direct*"/"
68  files = readdir(direc)
69
70  cluster_sizes = Array(Int64, (20, 2))
71  mol_count = Array(Float64, 20)
72  count_file = 1
73
74  for (ii, file) in enumerate(files)
75      words = ["inicial", "final"]
76      is_word = map((x) -> contains(file, x), words)
77      forbidden_words = ["levos", "dextros", "energies", "histograma", "grafica", "param
            ", "exceso", "tamano"]
78      is_forbidden_word = map((x) -> contains(file, x), forbidden_words)
79
80       # Escoger archivos con palabras .^especiales" para graficar
81      if true in is_word && !(true in is_forbidden_word)
82          try
83              # Graficar sistemas
84              sys_data = sign(readdlm(direc*file)) #Con el sign escoge por quiralidad
85              nmols = length(sys_data)
86              side = floor(Int64, sqrt(nmols))
87              # Contar clusters, guardar datos (que podría ir en el programa de la simulación) y graficar
88              clusters = count_cluster(sys_data)
89              cluster_list = []
90              for nclust in 1:maximum(clusters)
91                  sc = []
92                  for ii in 1:length(clusters)
93                      if clusters[ii] == nclust
94                          append!(sc, ii)
95                      end
96                  end
97                  append!(cluster_list, [sc])
98              end
99              size_of_clusters = sort(map(length, cluster_list), rev=true)#/nmols
100             writedlm(direc*"datos_histograma_"*file, size_of_clusters)
101
102             histogram = zeros(Int64, nmols, 2)
103             order_parameter = zeros(Float64, nmols, 2)
104             for jj in 1:nmols
105                 histogram[jj, 1] = jj
```

```
106                 for size_clust in size_of_clusters
107                     if size_clust == jj
108                         histogram[jj, 2] = histogram[jj, 2] + 1
109                     end
110                 end
111             end
112             writedlm(direc*"histograma_"*file, histogram)
113
114
115
116             # A partir de aquí, veré cuántos clusters hay de mas del 10 % del tamaño del sistema
117             medium_clusters = count(size_clust -> (size_clust > 0.05), size_of_clusters
                    /nmols)
118             big_clusters = count(size_clust -> (size_clust > 0.1), size_of_clusters/
                    nmols)
119             cluster_sizes[count_file, :] = [medium_clusters, big_clusters]
120             writedlm(direc*"tamano_clusters"*string(side)*".txt", cluster_sizes)
121
122             #Calcular exceso enantiomérico
123             der = count(chir -> (chir == 1), sys_data)
124             izq = nmols - der
125             ee = abs(der - izq)/Int64(nmols)
126             mol_count[count_file] = ee
127
128             count_file = count_file + 1
129         catch err
130             println(err)
131         end
132     end
133 end
134 writedlm(direc*"exceso_en.txt", mol_count)
135 end
136
137 Vector
```

Referencias

[1] Newman, M. E. J. y Barkema, G.T. (1999). *Monte Carlo Methods in Statistical Physics*. Oxford, Gran Bretaña: Oxford University Press.

[2] Chaikin, P. M. y Lubensky, T. C. (1995). *Principles of Condensed Matter Physics*. Cambridge, Gran Bretaña: Cambridge University Press.

[3] Onsager, L. (1944). Crystal Statistics. I. A Two-Dimensional Model with an Order-Disorder Transition. *Phys. Rev. 65*(3,4), 117-149.

[4] Eliel, E. L. (Ed.). (1993). *Stereochemistry of organic compounds*. Estados Unidos: John Wiley and Sons, Inc.

[5] Cahn, R. S., Ingold, C. y Prelog, V. (1966). Specification of Molecular Chirality. *Angew. Chem. Int. Ed. Engl., 5*, 385-415. doi: 10.1002/anie.196603851

[6] Nelson, D.L. (Ed.). (2017). *Principles of Biochemistry* Estados Unidos: Macmillan Learning.

[7] Blackmond, D. G. (2010). The origin of biological homochirality. *Cold Spring Harb. Perspect. Biol., 2*(5). doi: 10.1101/cshperspect.a002147

[8] Vargesson, N. (2015). Thalidomide-induced teratogenesis: History and mechanisms. *Birth Defects Research, 105*(2), 140-156. doi: 10.1002/bdrc.21096

[9] *Thalidomide - DrugBank*. Canadá: Drugbank.ca. Recuperado de: `https://www.drugbank.ca/drugs/DB01041`

[10] Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H. y Teller, E. (1953). Equation of State Calculations by Fast Computing Machines. *The Journal of Chemical Physics, 21*(6). Recuperado de `https://doi.org/10.1063/1.1699114`

[11] Frenkel, D. (Ed.). (2002). *Understandig Molecular Simulation: From Algorithms to Applications*. San Diego, Estados Unidos: Academic Press.

[12] Allen, M. P. (Ed.). (1987). *Computer simulations of liquids*. Oxford, Gran Bretaña: Oxford University Press.

[13] Andelman, D. y De Gennes, P.-G. (1988). Chiral discrimination in a Langmuir monolayer. *C. R. Acad. Sci., Ser. III, 307*(3), 233-237.

[14] Huckaby, D. A., Pitiş, R. y Shinmi, M. (1994) Existence of Enantiomeric Phase Separation in a Three-Dimensional Lattice Gas Model. *Journal of Statistical Physics, 75*(5,6), 981-995.

[15] Perusquía, R. A., Peon, J y Quintana, J. (2005). Two-dimensional model for mixtures of enantiomers, bent hard needles: a Monte Carlo simulation. *Physica A, 345*(1,2), 130-142. Recuperado de https://doi.org/10.1016/j.physa.2004.05.089

[16] Martínez-González, J. A., Pablo-Pedro, R., Armas-Pérez, J. C., Chapela, G. A. y Quintana-H, J. (2014). Chiral segregation of hockey-stick shaped particles in two dimensions. *RSC Adv., 4*(39), 20489-20495.

[17] Frenkel, D. (1993). Order through disorder: Entropy-driven phase transitions. En: Garrido, L. (Ed.). *Complex Fluids*. Lecture Notes in Physics, vol 415. Berlin, Alemania: Springer.

[18] Knuth, D.E. (1981). *The Art of Computer Programming* (Vol. 2). Addison Wesley.

[19] Conte, S.D (Ed.) (1980). *Elementary Numerical Analysis*. Estados Unidos: McGraw-Hill.

[20] Burden, R.L (Ed.) (Óscar Palmas, trad.). (2002). *Análisis numérico*. Ciudad de México, México: International Thomsom Editores. (Obra original publicada en 2001).

[21] IEEE Computer Society. (2008). *754-2008 - IEEE Standard for Floating-Point Arithmetic*. Recuperado de https://ieeexplore.ieee.org/document/4610935/

[22] Nie, Y.-Q., Huang, L., Liu, Y., Payne, F., Zhang, Y. y Pan, J.-W. 68 Gbps quantum random number generation by measuring laser phase fluctuations. *Review of Scientific Instruments, 86*(6), 063105. doi: 10.1063/1.4922417

[23] Zhang, L., Pan, B., Chen, G., Guo, L., Lu, D., Zhao, L., Wang, W. (2017). 640-Gbit/s fast physical random number generation using a broadband chaotic semiconductor laser. *Scientific Reports, 7*. 45900. doi: 10.1038/srep45900

[24] Haahr, M. *Introduction to Randomness and Random Numbers*. Dublín, Irlanda: Random.org. Recuperado de : https://www.random.org/randomness/

[25] The RAND Corporation. (1997). *A Million Random Digits*. Recuperado de : https://www.rand.org/pubs/monograph_reports/MR1418/index2.html

[26] Walker, J. *HotBits: Genuine Random Numbers*. Suiza: Fourmilab.ch. Recuperado de: https://www.fourmilab.ch/hotbits/

[27] *random(4) - Linux manual page*. Man7.org. Recuperado de : http://man7.org/linux/man-pages/man4/random.4.html

[28] Mackall, M. random.c - A strong random number generator. Git.kernel.org. Recuperado de https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/tree/drivers/char/random.c?id=refs/tags/v3.15.6#n52

[29] Matsumoto, M y Nishimura, T. (1998). Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator. *ACM Trans. on Modeling and Computer Simulation, 8*(1). 3-30. doi:10.1145/272991.272995

[30] Matsumoto, M. *Mersenne Twister Home Page.* Recuperado de `http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html`