Universidad Nacional Autónoma de México

FACULTAD DE CIENCIAS

Propiedades estadísticas de equilibrio térmico a partir de una red bidimensional con acoplamiento caótico

T E S I S

QUE PARA OPTAR POR EL GRADO DE:

FÍSICO

PRESENTA:

MANUEL ALEJANDRO ALDERETE LEZAMA

TUTOR:
DR. JUAN VALENTÍN ESCOBAR SOTOMAYOR

Ciudad Universitaria, CD. MX., 2022

1. Datos del alumno
Alderete
Lezama
Manuel Alejandro
55 54 57 34 32
Universidad Nacional Autónoma de México
Facultad de Ciencias
Física
313656413

2. Datos del tutor
Dr
Juan Valentín
Escobar
Sotomayor

3. Datos del sinodal 1
Dr
Raúl Patricio
Esquivel
Sirvent

4. Datos del sinodal 2
Dr
Ricardo Atahualpa
Solórzano
Kraemer

5. Datos del sinodal 3
Dr
Carlos
Málaga
Iguiñiz

6. Datos del sinodal 4
Dr
Giuseppe
Pirrucio

7. Datos del trabajo escrito
Propiedades estadísticas de equilibrio térmico a partir de una red bidimensional con acoplamiento caótico
188 p.
2022

Resumen

La presente tesis es un trabajo que busca encontrar ergodicidad, balance detallado y una distribución de Boltzmann en una rejilla de mapeo acoplado que presenta caos espacio-temporal, conforme a los resultados reportados en [5]. Se presentan los métodos computacionales usados para reproducir dichas observaciones. El estudio contempló sistemas con diferentes longitudes, acoplamientos y tamaños de grano (para las pruebas de balance detallado y distribución de Gibbs). Las simulaciones comprobaron que las rejillas de mapeo caótico acoplado efectivamente presentan dichas propiedades y dan sustento a una metodología prometedora para el análisis de sistemas fuera del equilibrio.

Agradecimientos

Aquí deseo plantear mis sinceros agradecimientos a todas las personas e instituciones que han contribuido en la realización del presente proyecto.

Agradezco a mi tutor, el Dr. Juan Valentín Escobar Sotomayor, su inmensurable apoyo durante mi Servicio Social y la elaboración de esta tesis. Su consejo y su calidad docente me ayudaron a afianzar la dirección de mi carrera y cultivar gradualmente múltiples habilidades y cualidades valiosas para mi desarrollo profesional y personal. Su paciencia y soporte fueron claves para superar los múltiples obstáculos que enfrentamos durante la investigación. Gracias a usted, hoy presento con orgullo y satisfacción el producto de nuestro esfuerzo.

Agradezco a los sinodales, el Dr. Raúl Patricio Esquivel Sirvent, el Dr. Ricardo Atahualpa Solórzano Kraemer, el Dr. Carlos Málaga Iguiñiz y el Dr. Guiseppe Pirruccio, el tiempo que dedicaron a la revisión del trabajo y sus perspicaces críticas, las cuales han fortalecido mis capacidades críticas y mejoraron de forma apreciable la calidad de la disertación.

Agradezco a los profesores cuyas clases atendí los conocimientos que compartieron conmigo y el sentido de responsabilidad, disciplina y perpetua curiosidad que inculcaron en mí a través de su excelente ejemplo. Asimismo, reconozco el arduo trabajo de sus respectivos ayudantes, cuya diligencia y disposición facilitaron en gran medida mi aprendizaje.

Agradezco a la Universidad Nacional Autónoma de México, la Facultad de Ciencias y el Instituto de Física por proveer la oportunidad de aprender más, ya fuera de increíbles libros, individuos admirables o traspiés y errores (más comunes de lo deseado).

Agradezco al Proyecto de Apoyo a Proyectos de Investigación e Innovación Tecnológica PAPIIT IT101820 y al Proyecto de Reactivación de Investigación y Docencia del Instituto de Física 2021 (PRIDIF21) el apoyo financiero proporcionado para la realización de este proyecto de titulación.

Agradezco a mi madre Elia por el gran amor que me has proporcionado todo este tiempo. Tu perseverancia, generosidad e inagotable optimismo me han inspirado a lo largo de los años a convertirme gradualmente en la mejor posible versión de mí, en un viaje repleto de enseñanzas y risas tan grandioso que no hallo las palabras para describirlo con justicia. Gracias a tu ayuda incondicional, he alcanzado todos mis logros.

Agradezco a mi tía Carmen por siempre apoyarme a alcanzar y aprovechar todas las oportunidades que han aparecido en mi vida. Aprecio mucho el afecto que me has deparado, las lecciones que me has dado y el tiempo que hemos convivido juntos, los cuales han contribuido significativamente en definir para bien mi identidad.

Agradezco a mi abuela Carmen el tierno y grato cariño que constantemente provees con tus palabras de aliento, que me animan a seguir adelante y perseguir mis metas.

Agradezco a mis tíos Manuel e Isabel, mis primos Christian y Jennifer y mis tías Lourdes y Cristina por el afecto y la confianza que han depositado en mí. Atesoro las agradables memorias que hemos formado juntos y me infunde de alegría la posibilidad de crear más en el futuro.

Agradezco a mis tíos Lourdes y Miguel su disposición a orientarme en mi desarrollo académico. Me han inspirado continuamente con su brillante carácter y su cariño también me ha impulsado a ser Físico.

Agradezco a los amigos que encontré en la Facultad, en particular a Luis Antonio, Aaron y Ruy, la cálida amistad que me ofrecieron, así como las risas y los consejos que compartimos en los altibajos de la carrera.

Índice general

Capítulo 1

Introducción

> El presente capítulo expone ejemplos de modelos y fenómenos característicos de sistemas fuera de equilibrio y contextualiza la tesis como uno de los múltiples trabajos en tal área de estudio enfocados en recobrar o adaptar las herramientas de física estadística para sistemas en equilibrio.

La física estadística estudia sistemas materiales compuestos por un gran número de elementos microscópicos y la relación entre el comportamiento de tales elementos y las propiedades macroscópicas de los sistemas que conforman. La cantidad de componentes de uno de estos sistemas es tan grande que resulta imposible o al menos ineficiente conocer las coordenadas en el espacio fase de cada partícula y usar tal conocimiento para determinar la dinámica del sistema. Sin embargo, dicha condición no representa un obstáculo insuperable sino la oportunidad de un cambio de paradigma; la física estadística aprovecha el hecho de que el número de partículas en los sistemas que analiza sea del orden de 10^{23} para implementar la teoría de la probabilidad en formular leyes y predicciones sobre ellos.

Una amplia y variada gama de sistemas han sido modelados exitosamente a través del formalismo de la física estadística, sin importar las disparidades en su escala o naturaleza; desde el comportamiento de gases ideales y materiales magnéticos hasta las propiedades de algunos tipos de estrellas pueden explicarse satisfactoriamente con dicha rama de la física. Sin embargo, la clave detrás de la extensa aplicabilidad de la física estadística radica en que tales sistemas son sistemas en equilibrio o exhiben muy pequeñas desviaciones respecto a tal estado cuyo estudio sólo requiere la inclusión de algunos términos lineales. La utilidad de la física estadística resulta entonces limitada e incluso nula para sistemas que se encuentran muy lejos del equilibrio; de hecho, cualquier investigación sobre sistemas de esta clase debe enfrentar además el reto de comprobar que estos efectivamente se hallan fuera del equilibrio, lo cual ha fomentado el diseño de inventivos métodos experimentales y sutiles reformulaciones de principios teóricos [6, 11].

La abundancia de sistemas fuera de equilibrio ha propiciado en gran medida su estudio formal y la peculiaridad de algunos de sus fenómenos característicos ha incentivado el desarrollo de enfoques que buscan complementar (e incluso adaptar) las herramientas de la física estadística para mejorar la comprehensión de estos sistemas.

Una de dichas perspectivas es la teoría de criticidad auto-organizada o teoría SOC (*Self-Organized Criticality*) propuesta por Per Bak y Kurt A. Wiesenfeld. Esta teoría pretende explicar la esencia de eventos catastróficos (como grandes terremotos, extinciones masivas o

crisis económicas) que diferentes sistemas complejos presentan [1]. En lugar de examinar el funcionamiento de las partes individuales que conforman un sistema bajo la suposición de que la magnitud de un evento en éste es proporcional a la de la perturbación que lo origina, la teoría SOC postula que algunos sistemas complejos tienden naturalmente a un estado crítico en el que cualquier acción, por pequeña que sea, puede producir respuestas tremendas en el sistema a través de las interacciones entre sus componentes[1]. Según esta teoría, la evolución de un sistema complejo con criticidad auto-organizada es distinta a la de sistemas en equilibrio afectados ocasionalmente por estímulos externos y los eventos en él, independientemente de su tamaño, no son el resultado de un mecanismo singular ni condiciones extraordinarias. Crucialmente, la teoría no precisa de un parámetro externo (como la temperatura en sistemas térmicos, por ejemplo) para explicar la llegada de un sistema a un estado crítico. Por ello, esta teoría ha sido usada existosamente para diseñar modelos para diversos sistemas complejos que presentan una criticidad por su propia cuenta o bien "auto-organizada".

Un ejemplo icónico de modelos de SOC es la pila de arena, cuyo comportamiento, estudiado inicialmente con simulaciones, ya ha sido observado experimentalmente. En este sistema, granos de arena son depositados individualmente en una superficie plana finita a un ritmo constante. Gradualmente, surge una pila que, pese al acomodo ocasional de los granos mediante pequeñas avalanchas, crece hasta alcanzar un punto crítico. En tal estado, la cantidad de partículas agregadas al sistema es equivalente (en promedio) a las que caen de la base, de forma tal que la altura y la pendiente de la pila de granos permanecen constantes. Las avalanchas en el sistema bajo esta configuración son reacciones en cadena detonadas por el deslizamiento inicial de un grano provocado por una inestabilidad en el sistema. Los consecuentes impactos de dicha partícula con otros elementos de la pila producen el movimiento de otros granos, los cuales a su vez experimentan sus propias colisiones con distintas partes del conjunto. Tal proceso entonces se ramifica por todo el sistema hasta que los granos de arena en movimiento alcanzan una posición estable o caen de la base. La señal singular de criticidad auto-organizada en una pila de arena es que la descripción anterior es válida para cualquier avalancha en el sistema, independientemente de su tamaño; en la teoría SOC, las catástrofes en sistemas complejos son como cualquier otro evento en ellos y ocurren aleatoriamente una vez que el sistema se encuentra en el punto crítico.

Así, leyes de potencia resultan ser excelentes aproximaciones de las distribuciones de eventos en sistemas complejos con criticidad auto-organizada; el tamaño de las avalanchas en pilas de arena, la energía liberada por sismos o el número de nacimientos o muertes en el Juego de la Vida de Conway obedecen funciones de tal forma (Figura 1.1). Dichas distribuciones representan una de las pruebas que evidencian criticidad auto-organizada en un sistema y su ubicuidad en varios campos de estudio exhibe claramente la utilidad del concepto.

[1]Tal condición se ve reflejada en la aparición de correlaciones de largo alcance entre las partes del sistema, conforme a [3], pág. 955

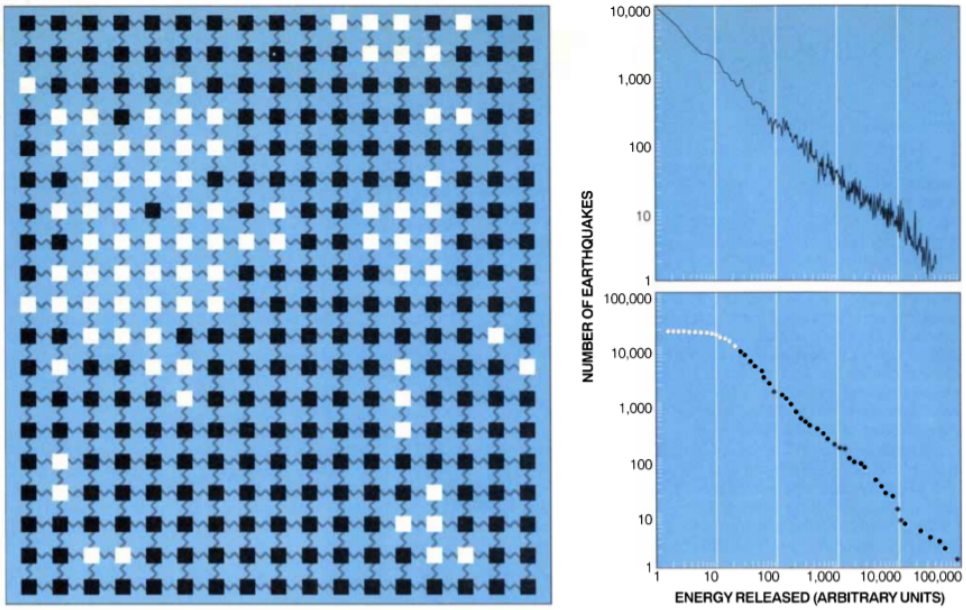


Figura 1.1: (Izq.) Esquema del modelo computacional del manto terrestre para simulaciones de sismos descrito en [1]. El modelo considera al manto como la unión de una placa flexible (conformada por bloques conectados a sus primeros vecinos mediante resortes) y una placa rígida. Al momento en que la fuerza ejercida por los resortes supera un umbral, un bloque se desplaza. Cada bloque en movimiento aparece de color blanco en el diagrama y un conjunto o *cluster* de ellos representa un sismo. La placa rígida ejerce fricción sobre los bloques en movimiento. (Der.) La gráfica superior muestra la distribución de energías construida a partir de 10,000 sismos simulados, mientras que la gráfica inferior resulta de mediciones de la energía de sismos en EE.UU. Ambas gráficas indican que el número de sismos está dado por una ley de potencias sobre su energía. Imagen procedente de [1].

Si bien la presencia de criticidad auto-organizada en una variedad impresionante de sistemas fuera de equilibrio es una prueba clara de su importancia, cabe resaltar que no es el único fenómeno de interés en este campo de estudio. Ante un estímulo externo variable en el tiempo, algunos sistemas pueden experimentar transiciones orden-desorden como el resultado de una respuesta coordinada entre sus componentes. Si bien tal acción conjunta es posible gracias a las interacciones locales entre los elementos del sistema, las transformaciones sobre él son controladas en parte por señales provenientes de su entorno (en contraste con los ejemplos de SOC).

Un ejemplo de esta condición es la alternancia en la concentración de calcio en células cardiacas, que ha sido descrita exitosamente como una transición orden-desorden en modelos sobre los mecanismos celulares que regulan el flujo de iones de calcio [7]. Alvarez-Lacalle *et al.* usan una rejilla de mapeo acoplado para explicar la alternancia de calcio como una consecuencia de las correlaciones de largo alcance del parámetro de orden del sistema, originadas por un estímulo periódico de alta frecuencia. Este análisis resulta de vital importancia, pues la alternancia de calcio en células cardiacas en un paciente es un indicio de que dicho individuo presenta un riesgo significativo de sufrir una arritmia letal.

No todos los estudios sobre sistemas complejos fuera de equilibrio se concentran en analizar este tipo de transiciones de fase a profundidad. Existen otros trabajos cuyo objetivo es mostrar que el equilibrio es en realidad tan sólo una de las muchas fases disponibles a ciertos sistemas y examinar detalladamente los parámetros que determinan a qué región del espacio de fases

corresponde cada configuración del sistema. Por ejemplo, en [8], Farmer argumenta la incorporación de los principios propios del estudio de sistemas complejos a la economía analizando un juego en el que la condición de equilibrio (idealizada en este campo de trabajo como una cualidad universal) no es un resultado factible en todos los casos.

Los fenómenos distintivos de sistemas fuera del equilibrio descritos previamente ameritan múltiples investigaciones para esclarecer su funcionamiento y describir aptamente su complejidad. Empero, existen otros esfuerzos vigorosos en otra dirección igualemente interesante: determinar la manera en que el formalismo de la física estadística puede adoptarse al estudio de sistemas fuera del equilibrio. De singular relevancia al presente trabajo son los estudios de sistemas con reglas de evolución temporal para sus componentes que mezclan un parámetro de acoplamiento entre vecinos cercanos con una variable regjida por una dinámica caótica [14]. Cabe destacar que no es evidente *a priori* que un sistema bajo dicha regla de evolución se comporte como un sistema en equilibrio térmico. En tal contexto, el trabajo de Egolf representa un avance significativo en los intentos por revolucionar la física estadística. En [5], se estudia una rejilla de mapeo acoplado o sistema CML (*Coupled Map Lattice*) y se reporta que dicho sistema, cuya regla de evolución temporal incorpora caos espacio-temporal, exhibe algunas propiedades típicamente halladas en sistemas en equilibrio. En particular, Egolf encuentra una escala de observación o "grano grueso" desde la cual la dinámica caótica desempeña el papel de ruido estadístico normalmente asociado a la temperatura. Más aún, muestra que tal rejilla de mapeo acoplado cuenta con las siguientes propiedades de sistemas termodinámicos en equilibrio:

1. Ergodicidad
2. Balance detallado
3. Distribución de probabilidad de Boltzmann

En otras palabras, el análisis de Egolf revela que el sistema CML (a una escala particular) se comporta como si sus elementos mantuvieran contacto con un reservorio de temperatura y la probabilidad de cada estado disponible a la rejilla dependiera de su energía, definida con un Hamiltoniano asignado concorde a las simetrías del sistema.

Tal hecho resulta verdaderamente notable porque constituye una prueba sólida de la existencia de un método para estudiar el comportamiento de sistemas cuyas leyes de evolución local, que incoporan factores estocásticos, no pretenden minimizar una energía total. Hallar un Hamiltoniano para estos sistemas permite calcular cantidades termodinámicas fundamentales y abre la puerta al estudio de sistemas no-térmicos utilizando la vasta maquinaria de la física estadística para, por ejemplo, determinar las condiciones necesarias que conducen a tales sistemas a un punto crítico y predecir transiciones de fase en ellos. Asimismo, el éxito de esta descripción posibilita entender el comportamiento de sistemas basados en reglas de interacción locales estocásticas a partir de reglas de minimización de cantidades globales.

El objetivo de la tesis es reproducir los resultados de Egolf y escribir códigos capaces de adaptar y estudiar el modelo de rejilla de mapeo acoplado a otros sistemas donde las interacciones entre sus elementos constitutivos tengan posiblemente una interpretación más concreta. En particular, presentamos un estudio de rejillas bidimensionales cuadradas de mapeo acoplado de diferentes tamaños a cuyos sitios $\vec{x}$ les corresponden variables $u^t_{\vec{x}}$ con la regla de actualización

$$u_{\vec{x}}^{t+1} = \phi(u_{\vec{x}}^{t}) + g \sum_{\vec{y}} \left[\phi(u_{\vec{y}}^{t}) - \phi(u_{\vec{x}}^{t})\right] \tag{1.1}$$

donde $\phi(u_{\vec{x}}^{t})$ es un mapeo caótico local definido como

$$\phi(u) = \begin{cases} -3u - 2 & -1 \leq u \leq -\frac{1}{3} \\ 3u & -\frac{1}{3} \leq u \leq \frac{1}{3} \\ -3u + 2 & \frac{1}{3} \leq u \leq 1 \end{cases} \tag{1.2}$$

Las limitaciones del *hardware* que implementamos al desarrollar la tesis redujeron el ámbito de esta investigación a sistemas "pequeños" en comparación con los que aparecen en [5]. A pesar de tal complicación, determinamos que rejillas de mapeo acoplado definidas por la regla de actualización en la Ecuación 1.1 y el mapeo ϕ en la Ecuación 1.2 sí exhiben algunas de las propiedades de sistemas en equilibrio reportadas por Egolf. Hallamos ergodicidad en sistemas CML de longitudes $L \geq 32$ y comprobamos la presencia de balance detallado y la existencia de una distribución de Boltzmann para rejillas de mapeo acoplado " a grano grueso" asociadas a rejillas "finas" de longitud $L = 32$ y granos de tamaño $G = 8^2$.

Los códigos desarrollados, descritos con detalle en los Apéndices, están disponibles en `https://github.com/maal22/Tesis_Licenciatura`.

[2]Los granos constituyen subregiones de $G \times G$ sitios de un sistema CML con longitud $L > G$, descritos en el Capítulo 3.

Capítulo 2

Conceptos fundamentales

El presente capítulo describe conceptos básicos de Física Estadística relevantes a la tesis. Comienza con la introducción del Principio fundamental de la Física Estadística y las características principales de un sistema en equilibrio. Después, proceden las definiciones de ergodicidad, teorema H y balance detallado. Luego, sigue una presentación del ensamble de Gibbs junto con la función de partición. FInalmente, el capítulo concluye con una breve descripción de transiciones de fase.

2.1. Principio fundamental de la física estadística

La siguiente sección está basada principalmente en [17], p. 57.

El **microestado** de un sistema corresponde al conjunto de valores de las coordenadas o variables asociadas a cada uno de los componentes de dicho sistema o estado cuántico, mientras que un **macroestado** ofrece una descripción más general. Por ejemplo, en el caso de un material paramagnético, un microestado queda determinado al conocer la orientación de cada uno de sus dipolos respecto al campo magnético externo aplicado. En cambio, definir un macroestado de dicho sistema únicamente requiere conocer el número de dipolos paralelos (o antiparalelos) a tal campo magnético.

Los sistemas térmicamente aislados son aquellos que no pueden intercambiar energía con su entorno y cuentan consecuentemente con una energía total fija. El comportamiento de sistemas de esta clase dispuestos en equilibrio térmico puede ser fácilmente estudiado bajo la suposición del llamado **principio fundamental de la física estadística**. Tal postulado establece que

Todos los microestados de un sistema aislado en equilibrio térmico son igualmente probables.

Al considerar verdadera esta suposición, el macroestado más probable en un sistema con las propiedades mencionadas previamente es aquél asociado con el mayor número de microestados distintos. Por ejemplo, para un material paramagnético conformado por cinco dipolos, estos macroestados corresponderían a la presencia de 2 o 3 dipolos paralelos al campo magnético externo aplicado al sistema, puesto que cada uno de ellos sería producido por 10 diferentes microestados.

Bajo el principio fundamental, las diferencias de probabilidad entre los distintos macroesta-

dos disponibles a un sistema aislado en equilibrio surgen de las diferencias en el número de microestados detrás de ellos, no en propiedades distintivas de ciertos microestados "atractores" que favorezcan su existencia; las transiciones entre todos los microestados del sistema son aleatorias e igualmente probables.

El principio fundamental es bastante intuitivo; ante la ausencia de una preferencia hacia algún microestado particular, es natural suponer que todos los microestados presentan la misma probabilidad. Sin embargo, esto no asegura que todos los microestados puedan ser registrados, puesto que el número de microestados de un sistema que pueden apreciarse en la práctica depende de cuánto tiempo se mantenga en observación tal sistema. De hecho, la numerosa cantidad de elementos en sistemas reales vuelve el número de microestados tan grande que es imposible detectar en tiempos razonables (experimentalmente hablando) al menos una vez todos los microestados a los que puede acceder el sistema. Al contemplar el principio fundamental, suponemos que la muestra recolectada de microestados es una muestra representativa.

2.2. Ergodicidad

El desarrollo presentado en esta sección procede de [15], pp. 583-585.

En un ensamble bajo un régimen estacionario, las variables $y^{(k)}(t)$ en sistemas k perteneciente a dicho ensamble pueden evaluarse a cualquier tiempo t y reproducir el mismo ensamble. Si tales sistemas cuentan adicionalmente con **ergodicidad**, la variable $y^{(k)}(t)$ accede a todos los valores disponibles a ella en el transcurso de un periodo con duración 2θ suficientemente larga. Consecuentemente, una colección de M intervalos (con una duración 2θ cada uno[1]) procedentes de la evolución temporal de un sistema constituye una muestra representativa del ensamble asociado a tal sistema[2].

La ergodicidad de un sistema se ve reflejada en la equivalencia entre dos tipos de promedios. Uno de ellos, el **promedio del conjunto**, es efectuado para una variable aleatoria sobre un ensamble a un tiempo dado y está definido como

$$\langle y(t) \rangle = \frac{1}{N} \sum_{k=1}^{N} y^{(k)}(t) \tag{2.1}$$

donde $y^{(k)}(t)$ representa el valor que adopta la variable y al tiempo t en el sistema k, dado un ensamble de N sistemas.

El otro promedio, en cambio, consiste en el **promedio temporal** de una variable aleatoria en un sistema particular sobre un periodo de larga duración y es equivalente a

$$\{ y^{(k)}(t) \} = \frac{1}{2\theta} \int_{-\theta}^{\theta} y^{(k)}(t + t') dt' \tag{2.2}$$

donde $\theta \to \infty$.

[1] Con una magnitud tan grande, la condición de ergodicidad implica que el comportamiento de $y^{(k)}(t)$ en cada intervalo es independiente de los demás.

[2] Asumimos adicionalmente que $M \gg 1$.

Las operaciones asociadas a estos dos promedios conmutan entre sí:

$$\left\langle \{y^{(k)}(t)\} \right\rangle = \frac{1}{N}\sum_{k=1}^{N}\left[\frac{1}{2\theta}\int_{-\theta}^{\theta} y^{(k)}(t+t')dt'\right] = \frac{1}{2\theta}\int_{-\theta}^{\theta}\left[\frac{1}{N}\sum_{k=1}^{N} y^{(k)}(t+t')\right]dt'$$
$$= \frac{1}{2\theta}\int_{-\theta}^{\theta}\left\langle y^{(k)}(t+t')\right\rangle dt' = \{\langle y(t)\rangle\} \tag{2.3}$$

Conforme a la discusión anterior, la presencia de ergodicidad en un ensamble estacionario implica que el promedio temporal de una variable aleatoria sobre un largo intervalo 2θ no depende del tiempo t en que se calcula. De hecho, tal atributo también asegura que dicho promedio es el mismo para cualquier sistema en el ensamble:

$$\{y^{(k)}(t)\} = \{y\} \tag{2.4}$$

Análogamente, el promedio de conjunto sobre un ensamble estacionario es independiente del tiempo, de modo tal que

$$\langle y(t)\rangle = \langle y\rangle \tag{2.5}$$

Al calcular el promedio de conjunto $\left\langle \{y^{(k)}(t)\} \right\rangle$, la Ecuación 2.4 implica que

$$\left\langle \{y^{(k)}(t)\} \right\rangle = \frac{1}{N}\sum_{k=1}^{N}\{y^{(k)}(t)\} = \frac{1}{N}\sum_{k=1}^{N}\{y\} = \{y\}\frac{1}{N}\sum_{k=1}^{N} 1 = \{y\}$$

o bien

$$\left\langle \{y^{(k)}(t)\} \right\rangle = \{y\} \tag{2.6}$$

De forma similar, el promedio temporal $\{\langle y(t)\rangle\}$ puede reformularse (conforme a la Ecuación 2.5) como

$$\{\langle y(t)\rangle\} = \frac{1}{2\theta}\int_{-\theta}^{\theta}\langle y(t+t')\rangle\, dt' = \frac{1}{2\theta}\int_{-\theta}^{\theta}\langle y\rangle\, dt' = \langle y\rangle\,\frac{1}{N}\frac{1}{2\theta}\int_{-\theta}^{\theta} dt' = \langle y\rangle$$

Lo anterior implica que

$$\{\langle y(t)\rangle\} = \langle y\rangle \tag{2.7}$$

Gracias a la conmutatividad de las operaciones asociadas a las dos clases de promedios (Ecuación 2.3), las ecuaciones 2.6 y 2.7 indican que un ensamble ergódico estacionario cumple que

$$\langle y\rangle = \{y\} \tag{2.8}$$

2.3. Teorema H

La siguiente sección sigue el desarrollo planteado en [15], pp. 624-626.

Sea $P_r(t)$ la probabilidad de observar a un sistema en su estado r al tiempo t. Asumiendo un conjunto discreto de estados, dicha probabilidad satisface la siguiente condición de normalización:

$$\sum_r P_r(t) = 1$$

Las interacciones entre los elementos de un sistema producen transiciones entre los estados accesibles. La transición de un sistema desde un estado r a un estado s puede ser descrita por un tasa de probabilidad w_{rs}, equivalente a la probabilidad por unidad de tiempo de observar tal transición. Dichas tasas pueden ser simétricas, de forma tal que

$$w_{rs} = w_{sr} \tag{2.9}$$

La probabilidad $P_r(t)$ experimenta dos tipos de cambios ante la evolución temporal del sistema. Su valor puede incrementar con las transiciones desde cualquier otro estado s al estado r, pero también puede disminuir a causa de las transiciones en el sentido inverso. Por consiguiente,

$$\begin{aligned} \frac{dP_r}{dt} &= \sum_s w_{sr} P_s - \sum_s w_{rs} P_r \\ &= \sum_s w_{rs} \left(P_s - P_r\right) \end{aligned} \tag{2.10}$$

gracias a la simetría entre las tasas de probabilidad expresada en la Ecuación 2.9.

Sea H el valor promedio del logaritmo natural de la probabilidad P_r, es decir,

$$H = \langle \ln P_r \rangle = \sum_r (\ln P_r) P_r \tag{2.11}$$

Entonces, sigue que

$$\begin{aligned} \frac{dH}{dt} &= \frac{d}{dt}\left[\sum_r (\ln P_r) P_r\right] = \sum_r \left[\frac{dP_r}{dt} \ln P_r + \frac{dP_r}{dt}\right] \\ &= \sum_r \frac{dP_r}{dt} \left(\ln P_r + 1\right) \end{aligned} \tag{2.12}$$

Tomando en cuenta la Ecuación 2.10, la expresión anterior da lugar a

$$\frac{dH}{dt} = \sum_r \sum_s w_{rs} \left(P_s - P_r\right) \left(\ln P_r + 1\right) \tag{2.13}$$

El intercambio de los índices r y s no afecta la suma en la igualdad anterior, de modo tal que

$$\frac{dH}{dt} = \sum_r \sum_s w_{sr} \left(P_r - P_s\right) \left(\ln P_s + 1\right) \tag{2.14}$$

Suponiendo la simetría entre las tasas de transición (Ecuación 2.9), la suma de las ecuaciones 2.13 y 2.14 resulta en

$$\begin{aligned}
\frac{dH}{dt} &= \frac{1}{2} \sum_r \sum_s w_{rs} \left[(P_s - P_r)(\ln P_r + 1) + (P_r - P_s)(\ln P_s + 1)\right] \\
&= \frac{1}{2} \sum_r \sum_s w_{rs} \left[-(P_r - P_s) \ln P_r + (P_r - P_s) \ln P_s\right] \\
&= -\frac{1}{2} \sum_r \sum_s w_{rs} (P_r - P_s)(\ln P_r - \ln P_s)
\end{aligned} \tag{2.15}$$

El logaritmo $\ln x$ es una función creciente de x, es decir, si $x_1 > x_2$, entonces $\ln x_1 > \ln x_2$. Esto implica que $(P_r - P_s)(\ln P_r - \ln P_s) \geq 0$ si $P_r \geq P_s$. Como la tasa w_{rs} es positiva, se tiene que

$$\frac{dH}{dt} \leq 0 \tag{2.16}$$

El resultado previo recibe el nombre de **Teorema H**.

La igualdad en la expresión 2.16 ocurre si las proabilidades P_r y P_s son equivalentes entre sí para todos los pares de estados r y s con tasas de transición w_{rs} no nulas. Esto implica que

$$\frac{dH}{dt} = 0 \quad \text{si } P_r = C \quad \forall r \tag{2.17}$$

donde C es una constante independiente del estado del sistema.

En un sistema aislado fuera de equilibrio, múltiples variables cambian constantemente en el tiempo. El Teorema H (Ec. 2.17) revela que H disminuye gradualmente mientras las probabilidades P_r de cada estado accesible r sean diferentes entre sí, independientemente de sus valores iniciales. El decrecimiento de H continúa hasta que $\frac{dH}{dt} = 0$ y todos los estados disponibles al sistema sean igualmente probables. Tal condición se mantiene y el sistema permanece en equilibrio en tiempos posteriores, puesto que cualquier cambio en las probabilidades P_r resultaría en un incremento de H y dicha posiblidad queda descartada por la Ecuación 2.17.

2.4. Balance detallado

El desarrollo aquí presentado sigue [15], p. 384.

Este postulado afirma que la probabilidad de un proceso en un sistema aislado en equilibrio es equivalente a la probabilidad del proceso inverso. En este contexto, un proceso consiste en una transición del sistema de un estado r a un estado s y el proceso inverso se refiere al proceso temporalmente inverso, es decir, la transición del sistema de un estado s^* a un estado r^* (donde r^* y s^* representan los estados derivados de los estados r y s respectivamente al considerar la transformación $t \to -t$).

El principio de balance detallado nace de la reversibilidad ante una inversión temporal inherente a las leyes microscópicas que rigen las interacciones entre las partes de un sistema. Las tasas de probabilidad relacionadas con transiciones entre estados del sistema son calculadas con tales leyes y consecuentemente también exhiben dicha reversibilidad. Por consiguiente, las tasas de probabilidad w_{rs} y $w_{s^*r^*}$ satisfacen la siguiente expresión:

$$w_{s^*r^*} = w_{rs} \tag{2.18}$$

Siendo P_r la probabilidad de que el sistema se halle en el estado r, la tasa de ocurrencia W_{AB} de una transición desde un conjunto A de estados denotados por r a un conjunto adicional B de estados marcados con s queda determinada como

$$W_{AB} = \sum_r \sum_s P_r w_{rs} \tag{2.19}$$

Análogamente, la tasa de ocurrencia $W_{B^*A^*}$ del proceso inverso está dada por

$$W_{B^*A^*} = \sum_{s^*} \sum_{r^*} P_{s^*} w_{s^*r^*} \tag{2.20}$$

Conforme al principio fundamental de la Física Estadística, todos los estados accesibles de un sistema aislado en equilibrio gozan de la misma probabilidad, lo cual implica que $P_r = P$ para todo estado r. Entonces, sigue que

$$W_{B^*A^*} = P \sum_{s^*} \sum_{r^*} w_{s^*r^*} = P \sum_r \sum_s w_{rs}$$

de acuerdo a la Ecuación 2.18. Surge aquí el principio de balance detallado, cuya esencia queda estipulada en la siguiente igualdad:

$$W_{B^*A^*} = W_{AB} \tag{2.21}$$

2.5. Ensamble de Gibbs

El análisis presentado en esta sección está basado principamente en [13], pp. 91-96, e incluye algunas identidades procedentes de [9], pp. 37, 49-50, y [2], pp. 40-41.

Un grupo de sistemas idénticos aislados en equilibrio representa un **ensamble microcanónico**. De acuerdo al principio fundamental de la física estadística, los estados disponibles a los sistemas pertenecientes a dicho ensamble cuentan con la misma probabilidad. Siendo N el total de estados, tal probabilidad P_i es igual a

$$\mathcal{P}_i = \frac{1}{N} \quad \forall i \tag{2.22}$$

En cambio, el **ensamble de Gibbs** o **ensamble canónico** es una colección de sistemas en contacto con un baño térmico. En este caso, la energía de un sistema del ensamble no es constante, sino que cambia ante el flujo de calor entre éste y el baño térmico que lo rodea.

El conjunto conformado por un sistema A y su respectivo baño térmico B sí cuenta con una energía interna constante U_T tal que

$$U_T = U_A + U_B, \tag{2.23}$$

donde U_A y U_B simbolizan las energías internas del sistema y su baño térmico respectivamente.

Además, la entropía de dicha agrupación está dada por la Ecuación de Boltzmann:

$$S = k_B \ln(\Omega), \tag{2.24}$$

donde k_B es la constante de Boltzmann y Ω representa el número de estados disponibles a un sistema cualquiera. En el caso del conjunto compuesto por A y B (dotado de Ω_T estados diferentes), su entropía S_T es igual a $S_T = k_B \ln(\Omega_T)$.

El número de estados asociado a un sistema depende de la energía interna de dicho sistema. Considerando la Ecuación 2.23 y tomando U_T como una constante, la dependencia de Ω_T con respecto a U_A queda planteada como

$$\Omega_T(U_A) = \Omega_A(U_A) \times \Omega_B(U_T - U_A)^3 \tag{2.25}$$

A continuación, procedemos a determinar la distribución de probabilidad asociada a un elemento del ensamble canónico. Para ello, es conveniente fijar al sistema A en un estado i con un energía E_i. Bajo tal restricción, $\Omega_A(U_A) = \Omega_A(E_i) = 1$, lo cual implica que

$$\Omega_T(U_A) = 1 \times \Omega_B(U_T - U_A) \tag{2.26}$$

La temperatura T de un sistema queda definida en términos de su entropía S y su energía interna U mediante la relación

$$\frac{1}{T} = \left(\frac{\partial S}{\partial U}\right)_X \tag{2.27}$$

donde X representa una variable extensiva, como el volumen lo sería para un gas.
Considerando la igualdad previa y la Ecuación de Boltzmann (2.24) para la entropía del baño térmico B, sigue que

$$\frac{1}{k_B T_B} = \left(\frac{\partial \ln(\Omega_B)}{\partial U}\right)_X \tag{2.28}$$

Como la temperatura del baño térmico es una constante T, integrar la Ecuación 2.28 revela que

[3]Aquí entra en juego el llamado *Principio Multiplicativo*, el cual establece que el total de formas en que dos eventos i y ii pueden ocurrir es equivalente al producto entre la cantidad de maneras distintas en que tales eventos pueden suceder individualmente; si los eventos i y ii pueden presentarse de n_i y n_{ii} formas diferentes, hay $n_i \cdot n_{ii}$ modos en que ambos eventos acaecen. En este caso, los eventos corresponden a la adopción de un estado particular por parte del sistema A y el baño térmico B y las cantidades n_A, n_B son la cantidad de estados disponibles al sistema A y el baño B respectivamente, denotadas por Ω_A, Ω_B conforme a la notación planteada en la Ecuación 2.24.

$$\Omega_B = \gamma \exp\left[\frac{U_B}{k_B T}\right] \tag{2.29}$$

donde γ consiste en una constante de integración.

De acuerdo a la Ecuación 2.29, la Ecuación 2.26 queda reformulada como

$$\Omega_T(E_i) = \Omega_B(U_T - E_i) = \gamma \exp\left[\frac{U_T - E_i}{k_B T}\right] \tag{2.30}$$

El número total de estados disponibles al par sistema - baño térmico queda determinado en la siguiente ecuación:

$$N_T = \sum_j \Omega_T(E_j) = \gamma \exp\left[\frac{U_T}{k_B T}\right] \sum_j \exp\left[\frac{-E_j}{k_B T}\right] \tag{2.31}$$

donde la suma se efectúa sobre los estados j disponibles al sistema A.

La probabilidad $\mathcal{P}_i$ de que el sistema A se encuentre en un estado i con una energía E_i es equivalente a la probabilidad de que el conjunto del sistema y el baño térmico se encuentre en alguno de los estados consistentes con dicha situación. Debido a las restricciones impuestas sobre él, dicho conjunto se comporta como un elemento de un ensamble microcanónico y así, la probabilidad $\mathcal{P}_i$ es igual a la razón entre el número de estados del conjunto con $U_A = E_i$ y el total de estados disponibles N_T:

$$\mathcal{P}_i = \frac{\Omega_T(E_i)}{\sum_j \Omega_T(E_j)} = \frac{\exp\left[-E_i/k_B T\right]}{\sum_j \exp\left[-E_j/k_B T\right]} \tag{2.32}$$

La expresión anterior es la llamada **distribución de probabilidad de Boltzmann**. Si bien dicha función de probabilidad para el sistema A no incluye los microestados asociados al baño térmico que lo rodea, éste mantiene cierta influencia al fijar la temperatura T. A bajas temperaturas, únicamente los estados con las energías E_i más pequeñas tienen una probabilidad significativa de ser ocupados. Si la temperatura del baño térmico es más alta, la probabilidad de que el sistema A acceda a estados con energías mayores incrementa.

El baño térmico B puede ser modelado como un conjunto de $M - 1$ copias del sistema A, siempre y cuando M sea suficientemente grande para garantizar que la capacidad calorífica de la colección de réplicas sea tan alta que su temperatura permanezca constante. Entonces, el conjunto conformado por el sistema A y el baño térmico B consiste en un grupo de M sistemas idénticos en contacto térmico entre sí. El número de estados Ω_T disponibles a la agrupación de M sistemas (dado un conjunto de números de ocupación $\{n_i\}$[4]) es equivalente al total de formas que hay de disponer n_i de los M sistemas en cada estado i.

Aquí resulta conveniente recordar que el total de combinaciones $C(n, r)$ de n elementos en grupos de r ($r \leq n$) es equivalente a

$$C(n, r) = \frac{n!}{r!(n-r)!} \tag{2.33}$$

[4]El número de ocupación $0 \leq n_i \leq M$ denota al número de sistemas dispuestos en el estado i y cumple que $\sum_i^{\text{ínf}} n_i = M$.

En particular, el total de formas de fijar n_1 de los M sistemas mencionados previamente en el estado 1 es igual al número de combinaciones $C(M, n_1)$:

$$C(M, n_1) = \frac{M!}{n_1!(M - n_1)!} \tag{2.34}$$

Una vez completado este acomodo, el número de maneras distintas de tener n_2 de los sistemas restantes en el estado 2 está dado por el número de combinaciones $C(M - n_1, n_2)$:

$$C(M - n_1, n_2) = \frac{(M - n_1)!}{n_2!(M - n_1 - n_2)!} \tag{2.35}$$

De forma análoga, la cantidad de arreglos diferentes con n_i de $M - \sum_j^{i-1} n_j$ sistemas en el estado i queda definida como

$$C\left(M - \sum_j^{i-1} n_j, n_i\right) = \frac{(M - \sum_j^{i-1} n_j)!}{n_i!(M - \sum_j^{i-1} n_j - n_i)!} \tag{2.36}$$

Así, el número de estados Ω_T es el producto de las combinaciones individuales $C(M - \sum_j^{i-1} n_j, n_i)$, de modo tal que

$$\Omega_T = \frac{M!}{n_1!n_2!n_3!...} \tag{2.37}$$

La aproximación de Stirling para factoriales de números n muy grandes establece que

$$\ln(n!) \sim n \ln(n) - n \tag{2.38}$$

Asumiendo que M y todos los números n_i sean suficientemente grandes para considerar válida la expresión 2.38, sigue que

$$\begin{aligned} \Omega_T &= \frac{M^M / \exp[M]}{\prod_i (n_i^{n_i} / \exp[n_i])} \\ &= \frac{M^M \times \prod_i \exp[n_i]}{\exp[M] \times \prod_i n_i^{n_i}} \\ &= \frac{M^M \times \exp\left[\sum_i n_i\right]}{\exp[M] \times \prod_i n_i^{n_i}} = \frac{M^M}{\prod_i n_i^{n_i}} \end{aligned} \tag{2.39}$$

puesto que $\sum_i n_i = M$. Entonces, conforme a la Ecuación 2.39, la entropía S_T del ensamble puede formularse como

$$S_T = k_B \ln(\Omega_T) = k_B \left\{ M \ln(M) - \sum_i n_i \ln(n_i) \right\} \tag{2.40}$$

Al considerar que $M \ln(M) = \sum_i n_i \ln(M)$, resulta que

$$S_T = k_B \sum_i n_i \ln\left(\frac{M}{n_i}\right) = -k_B M \sum_i \left(\frac{n_i}{M}\right) \ln\left(\frac{n_i}{M}\right) \tag{2.41}$$

En el límite $M \to \infty$, la razón n_i/M marca la probabilidad de que el sistema A se encuentre en el estado i. Por ende, el valor promedio de la entropía de un sistema cualquiera en un ensamble canónico puede ser expresado como

$$S = \frac{S_T}{M} = -k_B \sum_i \mathcal{P}_i \ln\left(\mathcal{P}_i\right) \tag{2.42}$$

2.6. Función de partición

La descripción de la función de partición aquí presentada sigue el desarrollo en [13], pp. 93-94 y 96-97.

La **función de partición Z** está definida como

$$Z = \sum_j \exp\left[-\frac{E_j}{k_B T}\right] \tag{2.43}$$

donde la suma es realizada sobre todos los estados j disponibles a un sistema en contacto con un baño térmico. Z también puede ser formulada como una suma sobre los niveles de energía, de modo tal que

$$Z = \sum_n g_n \exp\left[-\frac{E_n}{k_B T}\right] \tag{2.44}$$

siendo g_n la degeneración de la energía E_n, equivalente al número de estados cuyas energías son todas equivalentes a E_n.

La función de partición funciona como una constante de normalización para las probabilidades $\mathcal{P}_i$ en un sistema de un ensamble canónico, de modo tal que

$$\begin{aligned} \sum_i \mathcal{P}_i &= \sum_i \frac{\exp\left[-E_i/k_B T\right]}{\sum_j \exp\left[-E_j/k_B T\right]} \\ &= \frac{1}{Z} \sum_i \exp\left[-\frac{E_i}{k_B T}\right] = 1 \end{aligned} \tag{2.45}$$

Empero, la importancia de Z es mucho mayor de lo que la Ecuación 2.45 sugiere. Conforme a la Ecuación 2.32, la probabilidad $\mathcal{P}_i$ cumple la siguiente igualdad:

$$\ln(\mathcal{P}_i) = -\frac{E_i}{k_B T} - \ln(Z) \tag{2.46}$$

Al sustituir la igualdad previa en la Ecuación 2.42, se obtiene que

$$S = k_B \sum_i p_i \left(\frac{E_i}{k_B T} + \ln(Z)\right) = \frac{\hat{U}}{T} + k_B \ln(Z) \tag{2.47}$$

donde $\hat{U}$ es el valor promedio de la energía interna del sistema, es decir, $\hat{U} = \sum_i \mathcal{P}_i E_i$.

Los términos de la Ecuación 2.47 pueden ser reordenados para dar

$$\hat{U} - TS = -k_B T \ln(Z)$$

Como el valor medio de la energía libre de Helmholtz (denotada por F) es igual a la parte izquierda de la ecuación anterior, resulta que

$$F = -k_B T \ln(Z) \tag{2.48}$$

Así, la función de partición Z opera como una conexión entre la descripción microscópica de un sistema mediante el análisis estadístico de sus estados y el estudio del mismo objeto desde un punto de vista termodinámico.

2.7. Transiciones de fase

La información en esta sección procede de [13], pp. 259-262, y [2], pp. 255-258, 263.

Las fases de un sistema son los estados disponibles al sistema con propiedades físicas y químicas uniformes; dos muestras macroscópicas representativas de un sistema dispuesto en una fase determinada contarían con características físicas (como la densidad para un gas o la resistividad para un metal) y composición química equivalentes. Cambios abruptos en dichas propiedades constituyen una prueba clara de una transición de fase. Las peculiaridades de tales transformaciones permiten distinguir al menos dos clases de ellas: las transiciones de fase primer orden y las transiciones de fase de segundo orden (o continuas).

Transiciones de fase de primer orden

Las transiciones de fase de primer orden ocurren entre dos puntos de equilibrio asociados a fases distintas que ocupan regiones separadas en el espacio termodinámico del sistema y coexisten en un punto particular de esta clase de transiciones. Dichos puntos de equilibrio corresponden a mínimos locales de un potencial termodinámico (como la energía libre de Gibbs), siendo el menor de ellos la fase estable del sistema. Los demás puntos representan fases metaestables, a las cuales el sistema puede acceder brevemente antes de que fluctuaciones en el potencial termodinámico lo conduzcan (eventualmente) al estado verdaderamente estable[5].

Un ejemplo de transiciones de fase de primer orden es la transición líquido-gas, ilustrada como el proceso *A*-*B*-*C* en la Figura 2.1. En el punto A, el mínimo de la energía libre de Gibbs G corresponde a la fase con un menor volumen (el estado líquido). A través de cambios en la temperatura y la presión, el sistema es conducido al punto B sobre la curva de coexistencia entre las fases líquida y gaseosa, donde G tiene el mismo valor en ambos estados. Finalmente, el sistema alcanza el punto C y termina en la fase gasesosa, la cual constituye el punto de equilibrio más estable del sistema tras la transición de fase.

[5]Existen varios sistemas en los que tal transformación no ocurre rápidamente y permanecen en las fases metaestables por tiempos largos, como el vidrio o el acero.

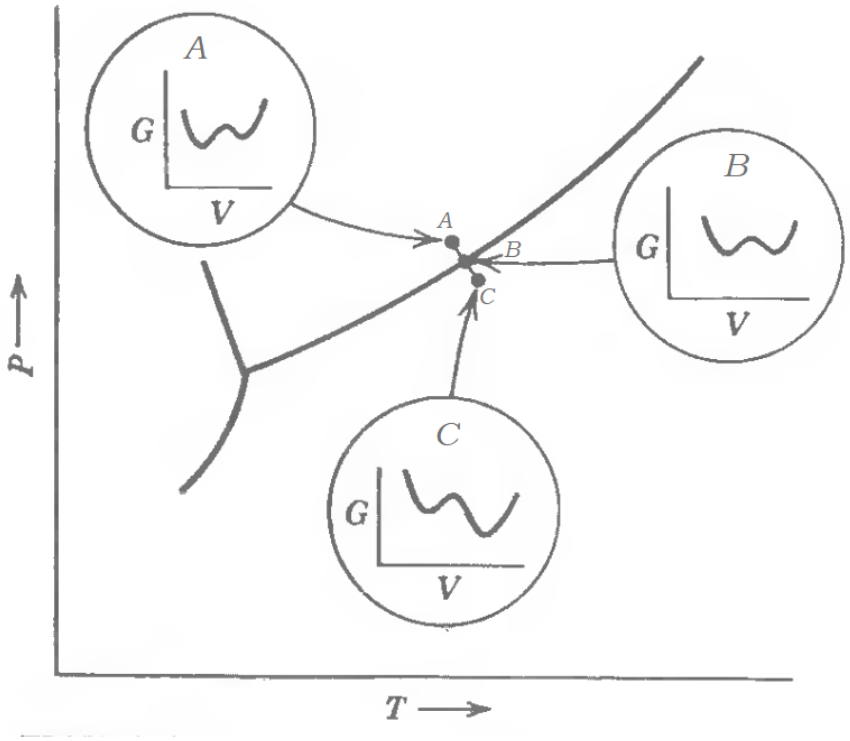


Figura 2.1: Transición de primer orden. Imagen adapatada de [2].

Adicionalmente, las transiciones de primer orden exhiben discontinuidades en diferentes potenciales molares. Por ejemplo, la diferencia de entropía entre las dos fases de una sustancia implica que dicha sustancia requiere un calor latente para cruzar la curva de coexistencia al experimentar una transición de fase.

Transiciones de fase de segundo orden

Las transiciones de fase de segundo orden en un sistema consisten en la separación de un punto único de equilibrio en múltiples estados. El proceso *A1-A2-Cr-B* en la Figura 2.2 es un ejemplo de estas transformaciones. La región del diagrama de fases con presiones y temperaturas superiores a las del punto crítico (marcado como Cr) corresponde a los fluidos supercríticos. Al acercar a un sistema en tal fase al punto crítico, el mínimo en la energía de Gibbs asociado a tal estado adquiere gradualmente la forma de una meseta hasta alcanzar el punto crítico y desarrollar dos mínimos locales. El máximo local que separa estos mínimos incrementa al conducir el sistema hacia el punto triple siguiendo la curva de coexistencia de fases (como en el punto B).

Una explicación más fructífera de las transiciones de segundo orden requiere la introducción de un nuevo concepto: el parámetro de orden. Éste consiste en una propiedad física del sistema que es distinta de cero bajo temperaturas menores a un valor crítico T_c y se vuelve nula a temperaturas superiores a T_c. Como su nombre lo indica, el parámetro de orden codifica una configuración u ordenamiento "especial" de los elementos del sistema a bajas temperaturas que desaparece con $T > T_c$; de ahí que las transiciones de segundo orden también sean conocidas como transiciones orden-desorden. Por ejemplo, la estructura cristalina de la aleación Cu-Zn es una red cúbica centrada. A bajas temperaturas, cada una de las redes que conforman el material está integrada por átomos del mismo elemento, de modo tal que cada átomo de cobre constituye el centro de un cubo con átomos de zinc en sus vértices (y viceversa). Al disponer

el sistema a una temperatura superior al punto crítico, tales redes ya no quedan conformadas exclusivamente por los mismos elementos y la aleación se vuelve "desordenada" (Figura 2.3).

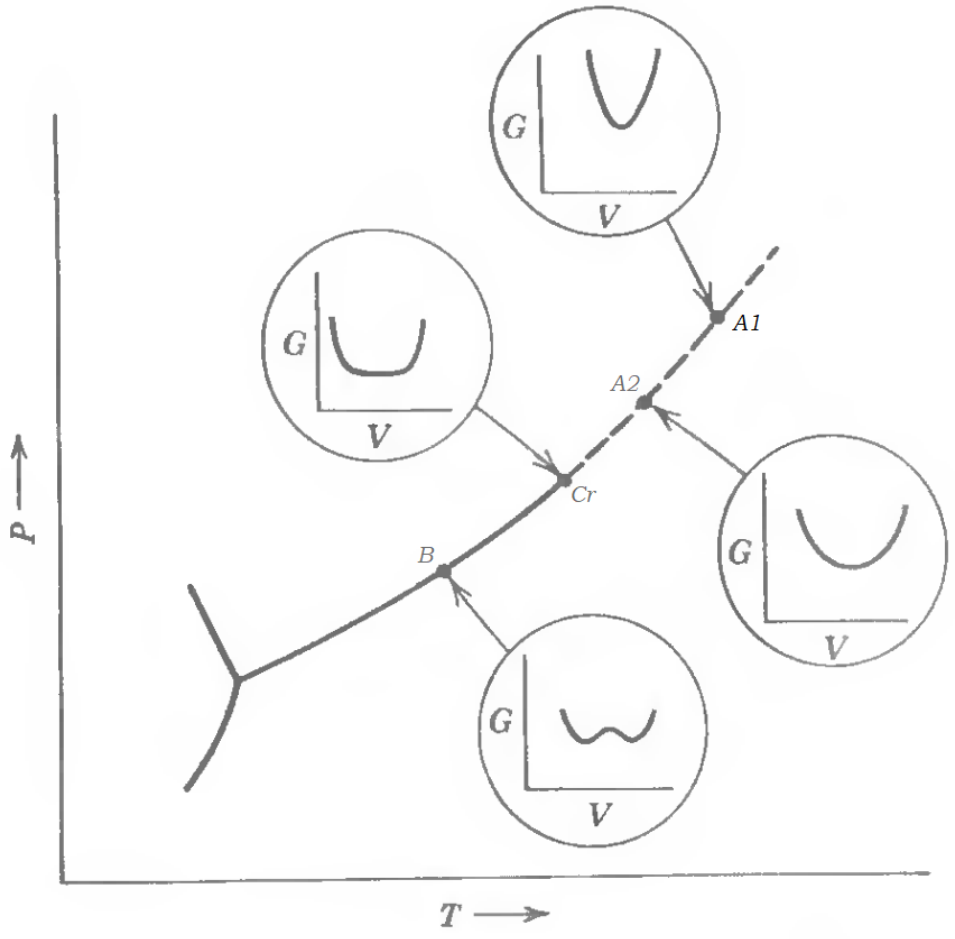


Figura 2.2: Transición de segundo orden. Imagen adaptada de [2].

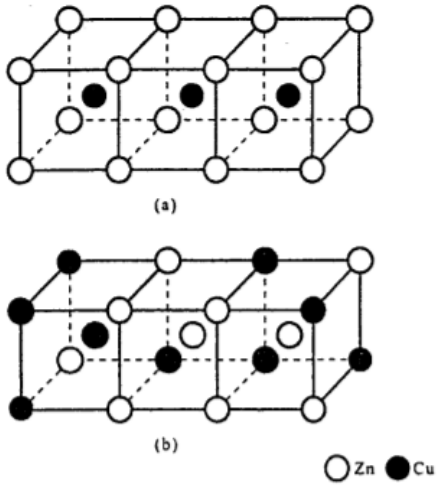


Figura 2.3: Transición orden-desorden en la aleación Cu-Zn, con las fases ordenada (a) y desordenada (b). Imagen procedente de [13].

2.8. Modelo de Ising

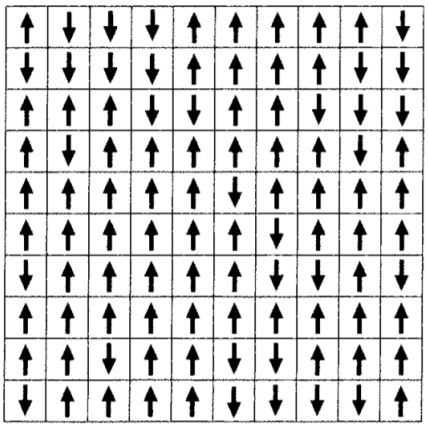
La información expuesta a continuación procede en su mayoría de [17], pp. 339-353, y [13], pp. 265-260.

El modelo de Ising es una representación idealizada de un material magnético que captura la interacción entre dipolos vecinos y exhibe transiciones de fase. Los dipolos en un sistema de esta clase únicamente pueden alinearse a lo largo de una dirección particular, ya sea de forma paralela o antiparalela (Figura 2.4a). La energía de una rejilla de Ising (libre de la influencia de un campo magnético externo) queda definida como

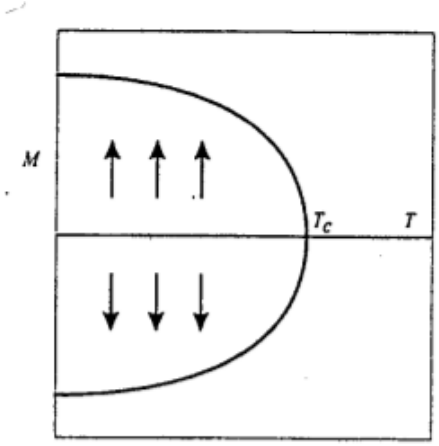
$$\mathcal{H} = -J \sum_{(i,j)} s_i s_j, \tag{2.49}$$

donde J denota el acoplamiento entre primeros vecinos, $s_i = \pm 1$ representa la magnitud de un dipolo i y la suma en la expresión anterior sólo contempla los primeros vecinos j de cada dipolo i. Si $J > 0$, el sistema es ferromagnético y los dipolos quedan todos alineados en un mismo sentido en el estado de mínima energía. En el caso contrario ($J < 0$), el modelo corresponde a un material antiferromagnético en cuyo estado de menor energía los dipolos vecinos apuntan a direcciones opuestas.

Cabe resaltar que un material ferromagnético de Ising, pese a su simplicidad, presenta una transición de segundo orden[6], con la magnetización del sistema (equivalente a la suma de los dipolos) como parámetro de orden. A bajas temperaturas, la magnetización es distinta de cero como consecuencia de una alineación uniforme de los susodichos dipolos. Una vez superada una temperatura crítica conocida como temperatura de Curie, el sistema se vuelve más desordenado, los dipolos apuntan en diferentes sentidos y la magnetización queda reducida a cero.



(a) Representación de un sistema de Ising bidimensional



(b) Transición orden-desorden característica del modelo de Ising bidimensional

Figura 2.4: Modelo de Ising. Imágenes procedentes de [13].

[6]Estrictamente, esto es válido para sistemas con una dimensionalidad mayor o igual a 2 y tal condición será asumida como cierta a lo largo de la siguiente discusión. El modelo de Ising unidimensional no experimenta una transición de esta clase.

metrías que distingue a las transiciones orden-desorden. La energía del sistema no cambia si la orientación del marco de referencia es invertida, es decir, H (Ecuación 2.49) es invariante ante la transformación $s_i \to -s_i$. Sin embargo, la magnetización del sistema sí puede ser modificada por esta inversión, siempre y cuando su valor sea distinto de cero. Por ende, la fase ordenada del modelo de Ising (observada a bajas temperaturas) pierde la simetría que presenta la fase desordenada.

La función de partición Z en el modelo de Ising queda definida como

$$Z = \sum_{\{s_i\}} \exp\left[\mathcal{H}\right] \tag{2.50}$$

donde $\{s_i\}$ representa el conjunto con todas las posibles configuraciones del sistema. Cada dipolo cuenta con dos posibles orientaciones, así que el número de elementos en la suma en la Ecuación 2.50 para un sistema con N dipolos equivale a 2^N. Por lo tanto, calcular las probabilidades de todos los estados disponibles al sistema (dadas por la Ecuación 2.32) resulta impráctico o incluso imposible para sistemas grandes. En la práctica, dichas probabilidades, así como otras propiedades termodinámicas del sistema, son obtenidas con un muestreo aleatorio de dichos estados. El método de muestreo más apropiado al modelo de Ising es el algoritmo de Metropolis, el cual implementa los factores de Boltzmann $\exp\left[-\mathcal{H}(\{s_i\})/k_BT\right]$ de cada estado $\{s_i\}$ al generar una muestra.

El algoritmo de Metropolis funciona de la siguiente manera:

1. Un dipolo s_i en el sistema es seleccionado al azar e invertido ($s_i \to -s_i$).
2. El cambio $\Delta\mathcal{H}$ en la energía provocado por tal cambio es calculado a partir de los valores del dipolo elegido y sus primeros vecinos.
   - Si $\Delta\mathcal{H} \leq 0$, entonces el cambio $s_i \to -s_i$ es aceptado y el sistema adquiere la configuración correspondiente.
   - Si $\Delta\mathcal{H} > 0$, el factor de Boltzmann correspondiente es computado, junto con un número aleatorio r entre 0 y 1.
     - En caso de que el número r supere el factor de Boltzmann, el cambio $s_i \to -s_i$ es aceptado.
     - En caso contrario, s_i recupera su valor original y el sistema permanece en el mismo estado.
3. El procedimiento anterior se repite con un nuevo dipolo s_j hasta obtener una muestra del tamaño deseado.

2.9. Caos

La presente sección está basada principalmente en [10], pp. 483, 491-494, [4], pp. 25-31, 45-57, 121-134, y [16], pp. 122-134.

En un sistema caótico, las trayectorias en el espacio fase que parten de puntos inicialmente cercanos entre sí describen movimientos enteramente distintos y una pequeña diferencia en las

condiciones iniciales no se refleja como tal en la evolución temporal del sistema (Figura 2.5). Ante nuestra incapacidad de efectuar mediciones con precisión absoluta, el futuro de un sistema caótico es incierto, incluso si las reglas que determinan su dinámica son bien conocidas.

Figura 2.5: Evolución temporal de un sistema "normal" (A) y un sistema caótico (B). Imagen procedente de [16].

La presencia del caos no está limitada a sistemas conformados por un vasto número de elementos. De hecho, algunos de los sistemas emblemáticos de tal fenómeno son mapeos iterativos unidimensionales, como la ecuación logística. Dicho mapeo está definido como

$$x_{n+1} = F(x_n) = Kx_n(1 - x_n) \tag{2.51}$$

donde $0 \leq x \leq 1$ y $K > 0$.

Modificar el valor del parámetro K produce cambios drásticos en el comportamiento del sistema, como lo ilustra la Figura 2.6. Dicha imagen, conocida como un diagrama de bifurcación, consiste en la gráfica de los puntos x_N obtenidos tras aplicar N veces consecutivas el mapeo logístico a un conjunto de diferentes condiciones iniciales $\{x_0\}$ para múltiples valores de K. Por ejemplo, las sucesiones $\{x_n = F^n(x_0)\}$ (denominadas órbitas de x_0 bajo F) tienden invariablemente a $x_N = 0$ con $K = 1$. mientras que $K = 3.2$ (o $K = 3.4$) produce órbitas periódicas que oscilan entre dos (o cuatro) puntos x_n distintos tras algunas iteraciones. Dado $K = 3.7$, la ecuación logística exhibe caos y las órbitas deambulan sin alcanzar un punto específico o un ciclo con periodo definido.

Figura 2.6: Diagrama de bifurcación de la ecuación logística. Imagen recopilada de [16].

Los casos descritos anteriormente para la ecuación logística permiten distinguir al menos dos tipos de puntos de singular importancia en mapeos caóticos:

- Un punto fijo x_f cumple que

  $$G(x_f) = x_f, \tag{2.52}$$

  dado un mapeo caótico $G(x)$. Tal condición implica que $G^n(x_f) = x_f \ \forall n$ (donde $G^n(x)$ denota la composición de la función $G(x)$ consigo misma n veces seguidas). Así, la órbita de x_f queda conformada únicamente por dicho punto.

- Un punto periódico x_p con periodo m es tal que

  $$G^m(x_p) = x_p \tag{2.53}$$

  siendo $G(x)$ un mapeo caótico. La órbita de x_p consiste entonces en una secuencia de números que se repiten tras m evaluaciones del mapeo $G(x)$.

2.9.1. Características principales del caos

Existen tres propiedades comunes entre sistemas caóticos:

1. **Densidad de puntos periódicos** - Dados los conjuntos X y Y tales que $X \subset Y$, se dice que X es denso en Y si para cualquier elemento $y \in Y$, existe un elemento $x \in X$ arbitrariamente cercano a y. En particular, el conjunto de puntos periódicos de un mapeo caótico $G(x) : [a, b] \to [a, b]$ son densos en el intervalo $[a, b]$.

2. **Transitividad** - Un sistema cuenta con transitividad si existe un punto z cuya órbita mantiene una distancia arbitrariamente pequeña con respecto a un par cualquiera de puntos x y y. Un mapeo $G(x) : [a, b] \to [a, b]$ es transitivo si hay una órbita densa en $[a, b]$ bajo tal función. Dicha condición garantiza hallar un elemento de semejante órbita cerca de cualquier punto en $[a, b]$.

3. **Sensibilidad a condiciones iniciales** - Ligeras discrepancias en las condiciones iniciales producen grandes cambios en la evolución de un sistema con este atributo. Por ejemplo, un mapeo $G(x)$ es sensible a condiciones iniciales si para cualquier par de puntos x y y (separados inicialmente una distancia ϵ arbitrariamente pequeña) existen un número real $\beta > 0$ y un natural k tales que la distancia entre $G^k(x)$ y $G^k(y)$ es mayor o igual a β.

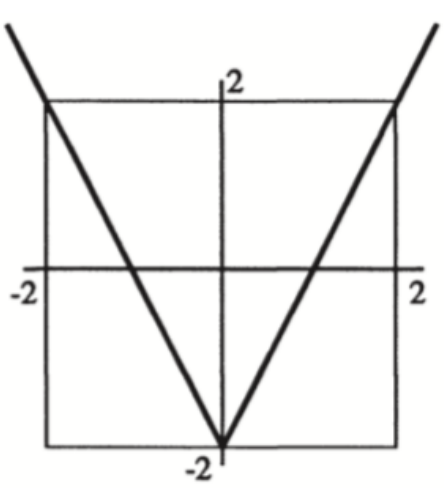
Las tres propiedades descritas anteriormente se encuentran en el mapeo $V(x) = 2|x| - 2$ en el intervalo $[-2, 2]$. La clave para comprobar tal afirmación es la forma de las gráficas correspondientes a $V^n(x)$ (como las que aparecen en la Figura 2.7). Dichas gráficas constan de 2^n partes, definidas en subintervalos con longitud 2^{2-n}. Cada uno de los segmentos de la gráfica consiste en una recta con pendiente $\pm 2^n$ que cubre el intervalo completo $[-2, 2]$.

La densidad de puntos periódicos implica que, dado un valor x cualquiera en el intervalo $[-2, 2]$, existe un punto x_p a una distancia arbitraria $\epsilon > 0$ que satisface la Ecuación 2.53. Conforme a las observaciones previas respecto a las gráficas de $V^n(x)$, existe un subintervalo $J(n) = [a_J, b_J] \subset [-2, 2]$ de longitud 2^{2-n} que contiene a x y mapea los puntos en su interior bajo $V^n(x)$ a $[-2, 2]$ de forma tal que $V^n(a_J) = -V^n(b_J) = \pm 2$. Aunado a esto, $V^n(x)$ es una función continua, así que el Teorema de Valor Intermedio ([19], pág. 124) asegura que existe un $x_p \in J(n)$ tal que $V^n(x) = x_p$.

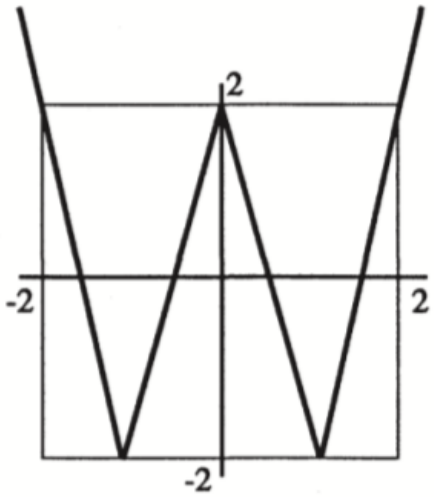
La existencia de órbitas densas bajo $V(x)$ en $[-2, 2]$ prueba la transitividad de dicho mapeo. Cada subintervalo $J(n)$ bajo $V^n(x)$ cubre el intervalo $[-2, 2]$ completo, lo cual indica que existen múltiples puntos cuya imagen es algún punto arbitrariamente cercano a un determinado valor x; de hecho, x en sí es parte de diferentes órbitas en $[-2, 2]$.

La sensibilidad a condiciones iniciales en el mapeo $V(x)$ queda clara al considerar (nuevamente) que la imagen de un subintervalo $J(n)$ es el intervalo $[-2, 2]$. Entonces, siempre es posible hallar dos puntos x y y dentro de un subintervalo $J(n)$ con una separación no nula $\epsilon \leq 2^{2-n}$ tales que $|V^n(x) - V^n(y)| \geq 2$.

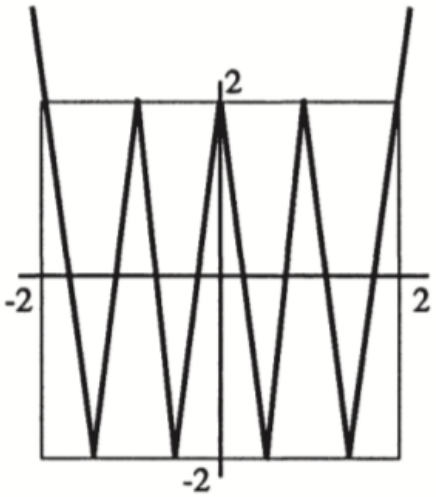
En particular, este atributo de sistemas caóticos puede visualizarse a través de series temporales (Figura 2.8 para el mapeo $V(x)$ o Figura 2.10 para la ecuación logística) o diagramas de trayectorias (Figura 2.9 para $V(x)$ y Figura 2.11 para la ecuación logística). La separación entre condiciones iniciales similares no se mantiene constante tras varias iteraciones y produce trayectorias muy diferentes para el sistema.



(a) Gráfica de mapeo $V(x)$



(b) Gráfica de $V^2(x)$



(c) Gráfica de $V^3(x)$

Figura 2.7: Gráficas del mapeo $V(x)$ y sus composiciones $V^2(x)$ y $V^3(x)$ en el intervalo $[-2, 2]$.

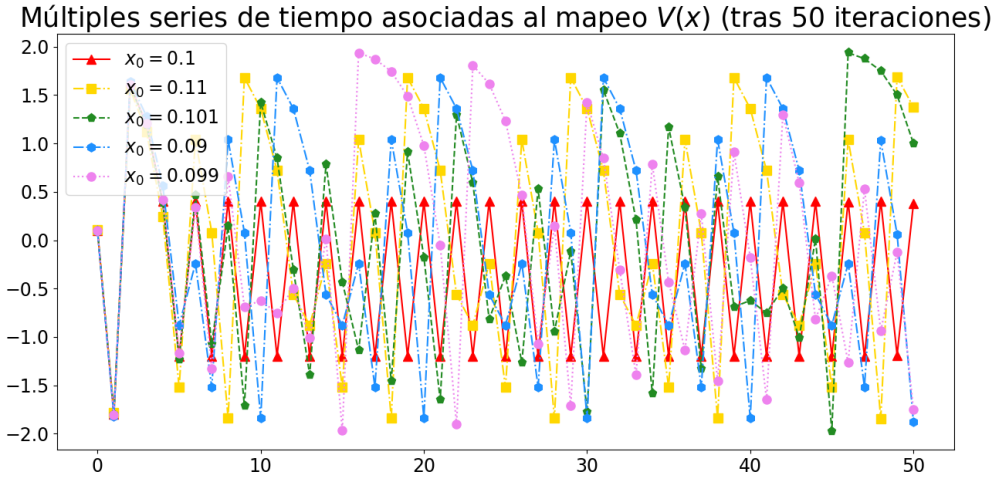


Figura 2.8: Serie temporal de mapeo $V(x)$ tras 50 iteraciones, considerando 5 condiciones iniciales x_0 diferentes.

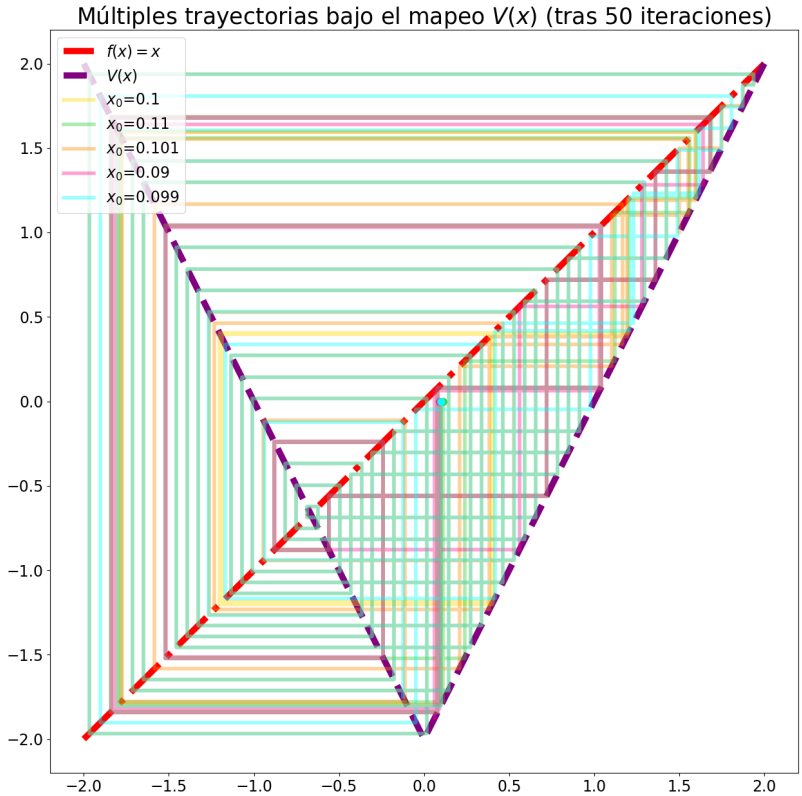


Figura 2.9: Diagrama de trayectorias bajo el mapeo $V(x)$ de cinco condiciones iniciales x_0 distintas tras 50 iteraciones.

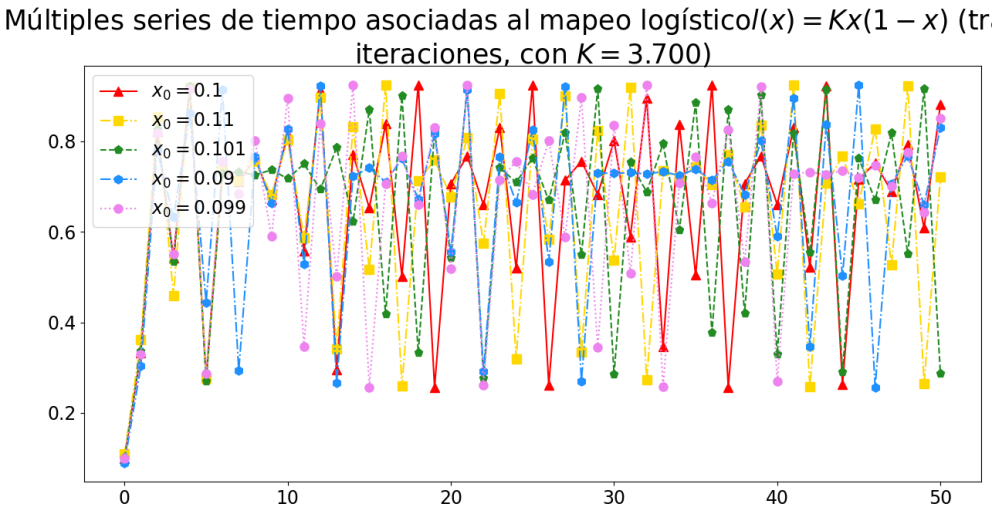


Figura 2.10: Serie temporal de ecuación logística tras 50 iteraciones, considerando 5 condiciones iniciales x_0 diferentes y $K = 3.7$.

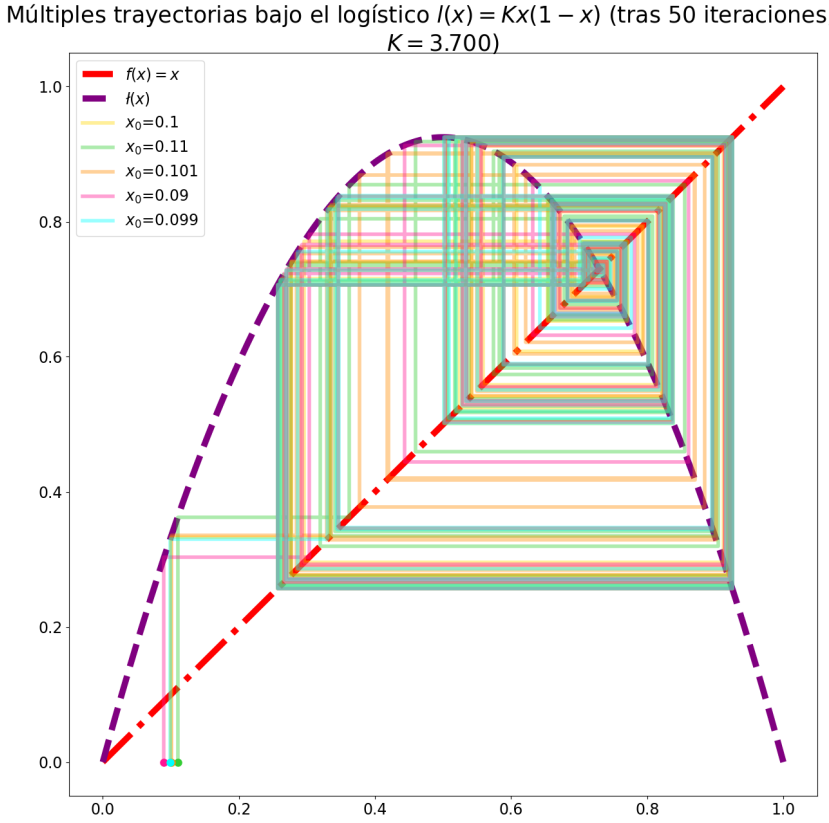


Figura 2.11: Diagrama de trayectorias bajo la ecuación logística de cinco condiciones iniciales x_0 distintas tras 50 iteraciones, con $K = 3.7$.

Capítulo 3

Rejilla de mapeo acoplado de Egolf

El presente capítulo expone el análisis por Egolf de una rejilla de mapeo acoplado con caos espacio-temporal [5]. En su trabajo, Egolf comprueba que dicho sistema exhibe ergodicidad incluso con un acoplamiento fuerte entre los sitios de la rejilla. También verifica que la dinámica del sistema es tal que existe una escala adecuada para el estudio de otras propiedades típicas de sistemas en equilibrio al examinar el comportamiento de los momentos centrales de promedios espaciales sobre subregiones de la rejilla del "espín" que caracteriza cada elemento de ella. Posteriormente, Egolf demuestra que el sistema a una escala específica cuenta con balance detallado e incluso una distribución de estados propia del ensamble canónico.

El caos espacio-temporal aparece en sistemas caóticos fuera del equilibrio con un gran número de grados de libertad. Tal condición resulta en comportamientos singulares de gran interés por su complejidad y la relación que guardan con la formación y el rompimiento de patrones en sistemas fuera del equilibrio ([3]). Una de las herramientas teóricas que facilitan su estudio son las rejillas de mapeo acoplado o sistemas CML *Coupled Map Lattice*, que consisten en rejillas cuyos sitios o elementos mantienen un acoplamiento con sus respectivos vecinos y evolucionan conforme a un mapeo específico (como la función logística). La simplicidad de los mapeos implementados en esta clase de sistemas vuelve el análisis de los mecanismos detrás del desorden o transiciones de fase mucho más sencillo.

En el artículo [5], Egolf observa que un sistema fuera de equilibrio con caos espacio-temporal exhibe a una escala adecuada múltiples propiedades distintivas de sistemas en equilibrio y propone que las herramientas de la física estadística designadas a sistemas en equilibrio pueden adaptarse al análisis de sistemas fuera de equilibrio. En particular, estudia una rejilla de mapeo acoplado bidimensional de $L \times L$ sitios con posiciones $\vec{x} = a\hat{x} + b\hat{y}$, donde a y b representan números enteros y $\hat{x}$ y $\hat{y}$ denotan vectores ortonormales. Al tiempo t, cada sitio de la rejilla está definido por un valor de la variable escalar $u^t_{\vec{x}}$, cuya regla de actualización es

$$u^{t+1}_{\vec{x}} = \phi(u^t_{\vec{x}}) + g \sum_{\vec{y}} \left[\phi(u^t_{\vec{y}}) - \phi(u^t_{\vec{x}})\right] \tag{3.1}$$

donde g marca la intensidad del acoplamiento espacial, $\vec{y}(\vec{x})$ denota los primeros vecinos del sitio $\vec{x}$ y el mapeo $\phi(u)$ está definido como

$$\phi(u) = \begin{cases} -3u - 2 & -1 \leq u \leq -\frac{1}{3} \\ 3u & -\frac{1}{3} \leq u \leq \frac{1}{3} \\ -3u + 2 & \frac{1}{3} \leq u \leq 1 \end{cases} \tag{3.2}$$

El mapeo $\phi(u)$ presenta caos espacio-temporal al menos para acoplamientos g dentro del intervalo $[0, 0.25]$ y experimenta una transición de fases para $g_c \approx 0.2054$ similar a la transición característica del modelo de Ising [14].

Egolf afirma que los atributos típicos de un sistema en equilibrio son apreciables en rejillas CML con evoluciones de duración tan larga como 10^6 iteraciones y ensambles conformados por hasta 256 sistemas, a lo largo de un gran rango de escalas (desde sistemas de 1×1 hasta sistemas de 1024×1024), asignando a cada sitio $u_{\vec{x}}^{t=0}$ un valor inicial aleatorio en $[-1, 1]$.

La primera cualidad de un sistema en equilibrio que Egolf detecta en una rejilla CML es ergodicidad, al menos para $0.18 < g < g_c = 0.2054$. Según lo planteado en la sección anterior, dicha condición implica la equivalencia entre el promedio temporal y el promedio sobre ensamble de una variable y en un sistema en equilibrio. Empero, no calcula ningún promedio para demostrar la ergodicidad de la rejilla, sino que opta por comparar las distribuciones de valores de la variable $u_{\vec{x}_0}^t$ en un sitio particular $\vec{x}_0$ obtenidas de la evolución de una sola rejilla y un ensamble de tales sistemas (Figura 3.1). La coincidencia entre las dos distribuciones garantiza que el sistema en cuestión sí es ergódico, al menos para cantidades asociadas a sitios individuales.

Adicionalmente, Egolf comprueba que un sistema suficientemente grande cuenta con *self-averaging* (dentro del mismo intervalo de acoplamientos g válidos). En otras palabras, muestra que la rejilla CML y un ensamble de ellas son equivalentes bajo la condición de que el sistema sea suficientemente amplio (Figura 3.1).

La presencia de balance detallado es la segunda característica de sistemas en equilibrio que las rejillas CML de Egolf exhiben. La dinámica de la Ecuación 3.1 implica una separación de escalas en el sistema y sugiere considerar en el estudio de balance detallado una escala "intermedia" en la cual el caos, apreciable a una escala microscópica, funja únicamente como ruido. Egolf confirma la validez del uso de semejante escala al analizar los momentos de las desviaciones de los promedios espaciales $\langle |u| \rangle_G$ (calculados sobre "subrejillas" de $G \times G$ sitios) con respecto al promedio $\langle |u| \rangle_{x,t}$. Las simulaciones de Egolf muestran que el caos en una rejilla CML se manifiesta como ruido blanco y que los momentos centrales de $\langle |u| \rangle_G$ cuentan con una distribución Gaussiana, puesto que los momentos pares (bajo una nueva norma [1]) son todos iguales y decaen como G^{-2}, mientras que los momentos impares tienden a cero (Figura 3.2).

[1] Los detalles de dicha "renormalización" se encuentran en el Apéndice B.

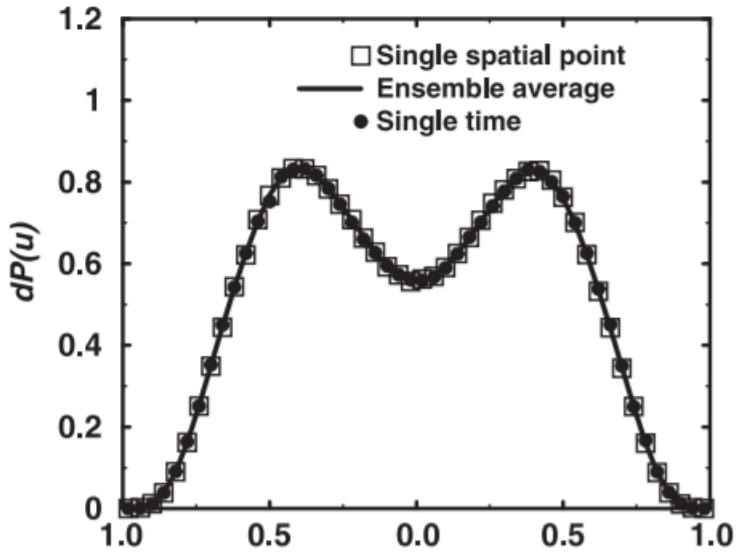


Figura 3.1: Distribuciones de probabilidad del observable $u^t_{\vec{x}_0}$ (con $\vec{x}_0 = (300, 500)$) en una rejilla de 512×512 tras 4×10^6 iteraciones para un sistema (cuadros) y un ensamble de ocho sistemas (línea), junto con la distribución de $u^{t_0}_{\vec{x}}$ para una rejilla de 1024×1024 sitios al tiempo $t_0 = 2 \times 10^5$. Imagen procedente de [5].

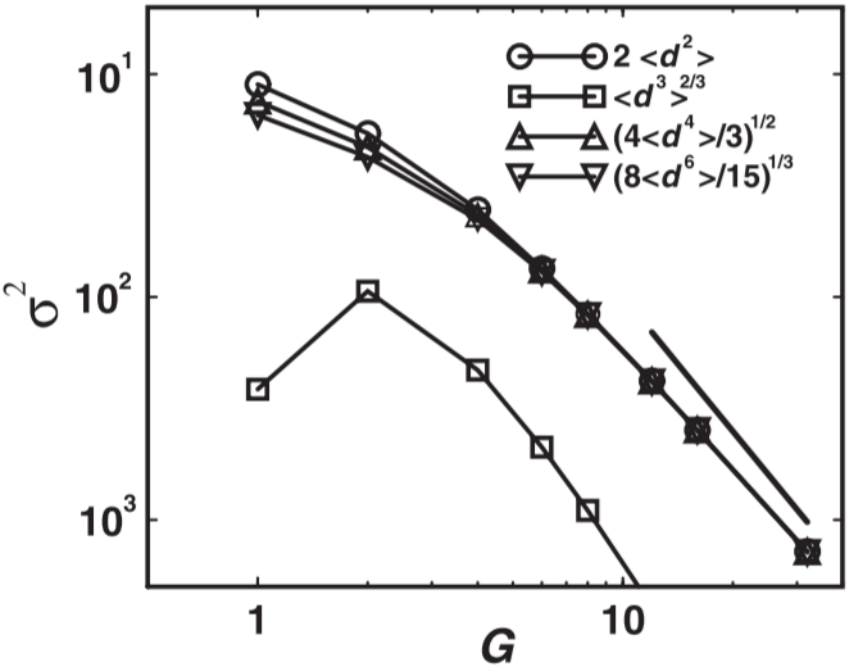


Figura 3.2: Comportamiento de segundo (círculos), tercero (cuadros), cuarto (triángulos) y sexto (triángulos invertidos) momentos de las desviaciones de $\langle |u| \rangle_G$ con respecto a su promedio $\langle |u| \rangle_{x,t}$. Imagen procedente de [5].

Las observaciones anteriores conducen a Egolf a estudiar balance detallado sobre rejillas "a grano grueso". A partir de un sistema CML de $GN \times GN$ sitios, Egolf construye un sistema CG-CML (*Coarse-Grained*) de $N \times N$ sitios. Cada uno de los sitios de esta nueva rejilla corresponden a regiones de $G \times G$ sitios del sistema CML y están asociados a una variable $\tilde{u}^T_{\vec{x}}$, definida como

$$\tilde{u}_{\vec{x}}^{T} = \text{sign}\left(\langle u_{\vec{y}}^{T}\rangle_{CG(\vec{x}),\Delta T}\right) \tag{3.3}$$

donde $CG(\vec{x})$ simboliza la colección de sitios $\vec{y}$ en la rejilla "original" que conforman la subrejilla de $G \times G$ asociada al sitio $\vec{x}$ de la rejilla a grano grueso.

Como el número de transiciones $T_{i\to j}$ de un estado i a un estado j que experimenta un sistema es equivalente a la tasa de ocurrencia W_{AB} (considerando $A = \{i\}$ y $B = \{j\}$), la condición de balance detallado expresada en la Ecuación 2.21 también puede plantearse como

$$T_{i\to j} = T_{j\to i} \tag{3.4}$$

Entonces, la diferencia $\Delta^{(i,j)} = T_{i\to j} - T_{j\to i}$ en un sistema dotado de balance detallado debería ser igual a cero para cualquier pareja de estados (i, j) entre los que ocurran transiciones. Sin embargo, la evolución del sistema CG-CML en una simulación está limitada a un tiempo finito durante el cual sólo acaecen $N^{(i,j)} = T_{i\to j} + T_{j\to i}$ transiciones para los pares de estados (i, j) disponibles, así que el cálculo de $\Delta^{(i,j)}$ a través de simulaciones únicamente se aproxima a cero.

Conforme al desarrollo incluido en el Apéndice C.2, la dependencia de $\Delta^{(i,j)}$ con el total de transiciones $N^{(i,j)}$ es tal que

$$\left\langle \left(\frac{\Delta^{(i,j)}}{N} \right)^{2} \right\rangle = N^{-1} \tag{3.5}$$

Las simulaciones de Egolf reproducen la relación 3.5 a lo largo de seis órdenes de magnitud (Figura 3.3), lo cual comprueba que el sistema CG-CML presenta balance detallado[2].

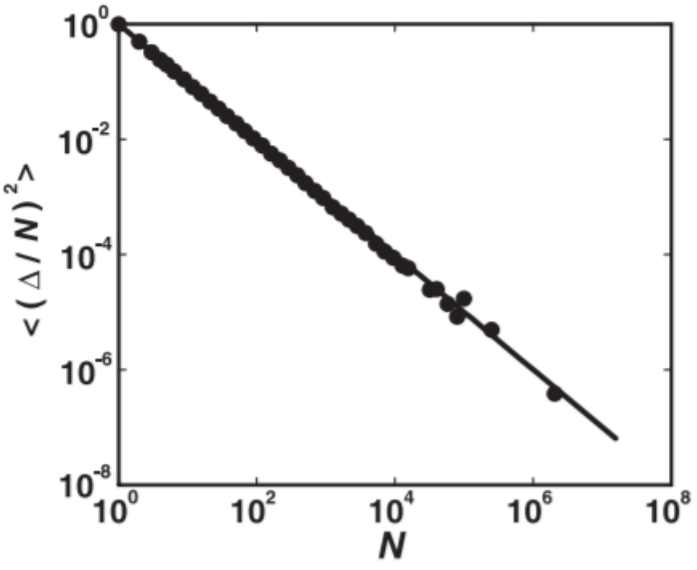


Figura 3.3: Relación entre el número de transiciones N y el valor esperado de $\langle(\Delta/N)^2\rangle$ para una simulación realizada con 10^6 pares de estados i y j dada una constante $g = 0.204$ en una rejilla de 64×64 sitios con un grano de tamaño $G = 16$ durante $T = 3.5 \times 10^8$ iteraciones de un ensamble de 256 sistemas. Imagen procedente de [5]

La presencia de balance detallado, así como la simetría de reflexión $f(u) = -f(-u)$ asociada a $\phi\,(u_{\vec{x}}^{t})$, sugieren que un Hamiltoniano con una simetría similar a la del modelo de Ising

(mencionada en la Sección 2.8) rige el comportamiento de la rejilla a grano grueso. Egolf considera un Hamiltoniano $\mathcal{H}$ definido como

$$\mathcal{H} = \sum_{\vec{x},\vec{y}} \alpha\,(d)\,\tilde{u}_{\vec{x}}^{T}\tilde{u}_{\vec{y}}^{T} \tag{3.6}$$

donde $\alpha(d)$ representa el acoplamiento entre los sitios $\vec{x}$ y $\vec{y}$ de la rejilla con una separación $d = |\vec{x} - \vec{y}|$ entre ellos.

Egolf verifica que la distribución de probabilidad de los estados S_i disponibles a la rejilla a grano grueso corresponde a una distribución de Boltzmann al estimar la probabilidad de cada estado a través de simulaciones y calcular los coeficientes $\alpha(d)$[3] ajustando a tales probabilidades una distribución de probabilidad $\mathcal{P}(S_i)$ definida como

$$\mathcal{P}(S_i) = \frac{\exp\{-\mathcal{H}(S_i)\}}{\sum_{S_i}\exp\{-\mathcal{H}(S_i)\}} \tag{3.7}$$

Nuevamente, las simulaciones de Egolf demuestran la validez del Hamiltoniano $\mathcal{H}(S_i)$ y la distribución de Gibbs correspondiente al reproducir la relación entre la probabilidad $\mathcal{P}(S_i)$ y dicho Hamiltoniano (Figura 3.4) con los coeficientes $\alpha(d)$ reportados en la Tabla 3.1, exhibiendo en un sistema caótico fuera del equilibrio una tercera propiedad típicamente hallada en sistemas en equilibrio[3].

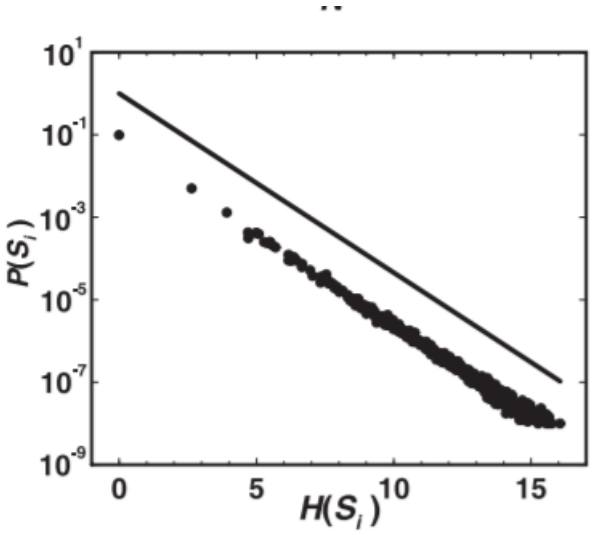


Figura 3.4: Relación entre la probabilidad de un estado S_i y su respectiva energía. Las condiciones de la simulación que produjo esta gráfica son las mismas que las de la simulación usada para la Figura 3.3. Imagen procedente de [5]

[3]Egolf únicamente contempla los cuatro primeros coeficientes al calcular la energía $\mathcal{H}(S_i)$, aunque no ofrece una explicación al respecto.

[3]Egolf reporta balance detallado y distribuciones de Gibbs en sistemas CG-CML de 4×4 sitios. Egolf no ofrece las dimensiones específicas de tales rejillas (a excepción del sistema asociado a las Figuras 3.3 y 3.4), pero afirma haber obtenido resultados consistentes para tamaños de grano $G \geq 8$.

$\alpha(1)$	$\alpha(\sqrt{2})$	$\alpha(2)$	$\alpha(\sqrt{5})$	$\alpha(\sqrt{8})$
0.685	0.100	-0.075	-0.049	-

Cuadro 3.1: Acoplamientos $\alpha(d)$ ajustados a distribución experimental de probabilidades $\mathcal{P}(S_i)$ de sistema CG-CML de 4×4 sitios con una evolución temporal de 3.5×10^8 iteraciones, elaborado a partir de una rejilla de mapeo acoplado de longitud $L = 64$, tamaño de grano $G = 16$ y acoplamiento $g = 0.204$

Egolf también nota que el comportamiento del coeficiente $\alpha(1)$ es similar al de acoplamientos en sistemas en equilibrio con simetría de Ising. Dado $g < g_c$, la rejilla se encuentra en la fase paramagnética y $\alpha(1)$ tiende a un punto fijo $\alpha(1) = 0$, correspondiente a temperatura $T \to \infty$. En cambio, con $g > g_c$, el sistema queda dispuesto en la fase ferromagnética y el coeficiente fluye a un punto $\alpha(1) \to \infty$, asociado a una temperatura igual a cero.

La importancia de que un Hamiltoniano describa la estadística de un sistema caótico fuera del equilibrio radica en que tal hecho implica que alguna función del acoplamiento g entre los elementos del sistema desempeña el rol de temperatura, lo cual podría permitir la descripción de transiciones de fase en sistemas fuera del equilibrio.

Capítulo 4

Resultados

> El presente capítulo describe las simulaciones realizadas con el propósito de reproducir las observaciones reportadas en [5] y exhibe un análisis detallado de los resultados obtenidos.

El objetivo principal de la tesis fue replicar exitosamente algunas de las aserciones por Egolf sobre la rejilla de mapeo acoplado o sistema CML estudiado en [5]:

1. El sistema CML exhibe ergodicidad.
2. Los momentos centrales de $\langle |u| \rangle_G$ muestran un comportamiento análogo al de ruido blanco.
3. Una rejilla a grano grueso o sistema CG-CML cuenta con balance detallado.
4. La distribución de estados de un sistema CG-CML es una distribución de Boltzmann.

En las simulaciones usadas para cumplir tal meta, el sistema CML es modelado como una gráfica `Graph` del paquete `networkx` de Python. Los sitios del sistema son definidos como nodos de dicha gráfica. Cada uno de ellos recibe un número entero $n(\vec{x})$ para marcar su posición $\vec{x}$ en la rejilla y cuenta con una propiedad `'u'` con un valor inicial aleatorio entre -1 y 1. Las distancias entre todos los posibles pares de nodos en la gráfica son calculadas bajo condiciones de frontera periódicas y almacenadas como los valores del atributo `'path distance'` de los bordes de la gráfica.

Las simulaciones implementan una función propia para determinar los primeros vecinos $\vec{y}(\vec{x})$ de un sitio particular $\vec{x}$ en la rejilla, puesto que la regla de actualización de $u^t_{\vec{x}}$ (Ecuación 1.1) contempla sumas sobre ellos. La aplicación de tal función sobre cada sitio en la rejilla permite agrupar en una lista el conjunto de primeros vecinos para todos los sitios en el sistema.

4.1. Caos

El primer análisis efectuado sobre los sistemas CML tenía por objetivo comprobar que el mapeo $\phi(u)$ (Ecuación 3.2) fuera un mapeo caótico. El estudio del mapeo $V(x)$ en la Sección 2.9 indica que $\phi(u)$ efectivamente exhibe tal propiedad. Análogamente al caso de $V(x)$, la gráfica de $\phi^n(u)$ queda conformada por 3^n segmentos de rectas con pendientes $\pm 3^n$ definidos en intervalos de longitud 3^{-n}. Por ende, podemos seguir los mismos razonamientos presentados en la Sección

2.9 para afirmar que $\phi(u)$ es un mapeo caótico, como también lo sugieren la serie temporal y el diagrama de trayectorias incluidos en las Figuras 4.1 y 4.2.

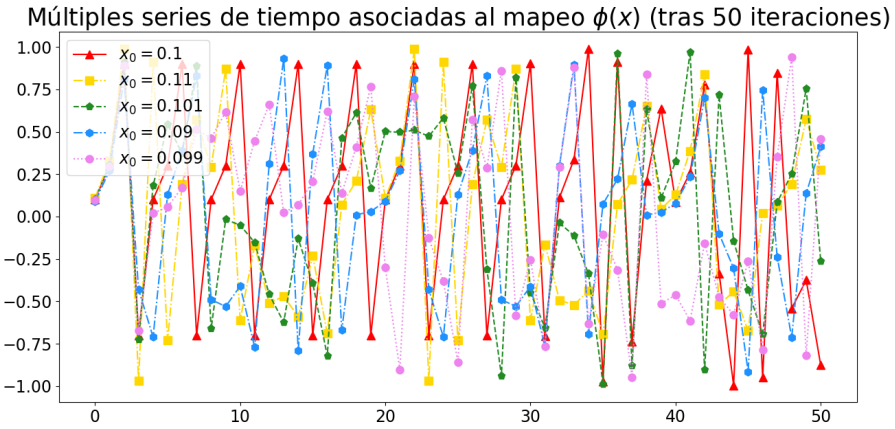


Figura 4.1: Serie temporal de mapeo $\phi(u)$ tras 50 iteraciones, considerando 5 condiciones iniciales u^{t_0} diferentes.

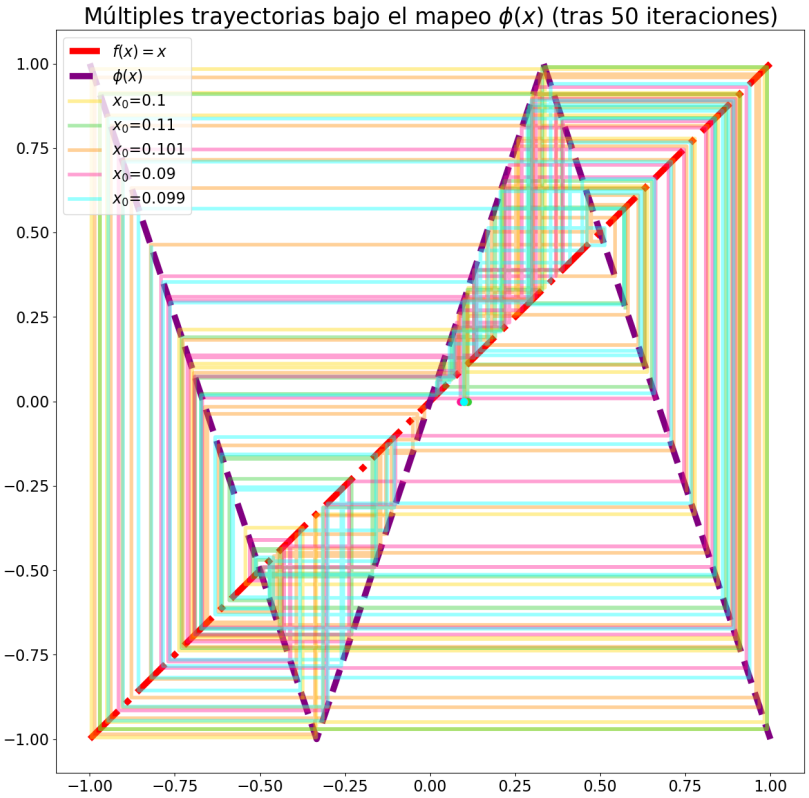


Figura 4.2: Diagrama de trayectorias bajo el mapeo $\phi(u)$ de cinco condiciones iniciales u^{t_0} distintas tras 50 iteraciones.

4.2. Ergodicidad

El código en el Apéndice D genera y compara distribuciones de probabilidad de la variable $u^t_{\vec{x}_0}$ considerando separadamente la evolución de una rejilla de mapeo acoplado y un ensamble de tales sistemas.

En los experimentos realizados para probar la presencia de ergodicidad del sistema, contemplamos diferentes valores para el acoplamiento entre sitios y la duración de un período transitorio.

4.2.1. Detalles Técnicos Computacionales sobre la prueba de Ergodicidad

El programa inicia seleccionando aleatoriamente un número natural s entre 0 y el valor de `sys.maxsize`[1] menos 1 (líneas 125-126 del código en Apéndice D) y solicitando al usuario los siguientes datos (127-133):

- Longitud L de la rejilla de mapeo acoplado
- Acoplamiento g entre los sitios del sistema CML
- Duración total y duración del período transitorio (*transient*) de la evolución temporal de la rejilla
- Número de sistemas N_{ens} que conforman el ensamble

Una vez escogido azarosamente el número de nodo $n(\vec{x}_0)$ de un sitio $\vec{x}_0$ arbitrario y fijado s como la semilla para el generador de números aleatorios[2] (136-137), la gráfica del sistema CML y la lista con los primeros vecinos de todos sus sitios son definidos (138-143).

Luego, el código utiliza la función `ev_temp` (73-119) para producir la evolución temporal de la rejilla bajo los parámetros establecidos, conforme a la regla de actualización dada por la Ecuación 1.1 y el mapeo $\phi(u)$ definido en la Ecuación 1.2 (145-147). Durante este proceso, el programa guarda el valor de `'u'` del nodo $n(\vec{x}_0)$ que adquiere en cada iteración del sistema. Una vez concluida la evolución de la rejilla, el arreglo con los valores de `'u'` en el sitio $\vec{x}_0$ es almacenado en un archivo de texto (149-151).

Después, el procedimiento anterior es repetido N_{ens} veces distintas, asginando nuevos valores de `'u'` a cada nodo de la gráfica antes de que ésta inicie su evolución temporal (156-162). Al finalizar el ciclo `for` en que esto ocurre, la matriz conformada por los arreglos de valores de `'u'` asociados a diferentes sistemas es "aplanada" en un arreglo unidimensional que es posteriormente guardado en otro archivo `.txt` (164-166).

El programa procede entonces a construir la gráfica de una rejilla de mapeo acoplado de $2L \times 2L$ sitios (con el mismo acoplamiento g) y su correspondiente lista con primeros vecinos de cada sitio en ella (170-176). Acto seguido, el código aplica la función `ev_temp` a dicha gráfica para inducir su evolución temporal sin conservar los valores de 'u' para el sitio $\vec{x}_0$ (178-180). Tras

[1] El valor proporcionado por `sys.maxsize` es equivalente a $2^{63} - 1$ para platafromas de 64 bits.

[2] En primera instancia, la hora actual del sistema desempeña el rol de la semilla del generador de números aleatorios y sirve para elegir al azar un nuevo valor s para la semilla. Tal natural s aparece después en los nombres de los archivos generados por la simulación para facilitar la reproducibilidad de los resultados.

acabar este proceso, los valores de `'u'` en cada sitio del sistema son recolectados en un arreglo y un archivo de texto adicional (181-187).

Finalmente, el programa construye histogramas con los tres arreglos definidos previamente y grafica los resultados (191-231).

4.2.2. Resultados de la prueba de Ergodicidad y *Self-averaging*

Estudiamos la condición de ergodicidad analizando rejillas de mapeo acoplado de 32×32 sitios con una evolución temporal de 2×10^5 iteraciones[3]. Contemplamos dos valores distintos para el acoplamiento entre los sitios del sistema: $g_1 = 0.195$ y $g_2 = 0.204$. También consideramos periodos transitorios de 10^3 y 10^4 iteraciones. Evaluamos la presencia de *self-averaging* en rejillas dotadas del doble de longitud ($L = 64$) y la misma duración para su evolución temporal, con acoplamientos y *transients* sujetos a las variaciones detalladas anteriormente.

Los resultados de las simulaciones para ergodicidad aparecen a continuación (Figuras 4.3-4.6). Cada Figura incluye tres pares de gráficas, cada par asociado a una semilla s distinta y un sitio $\vec{x}_0$ específico. Las gráficas en la columna izquierda (Gráficas (a), (c) y (e)) corresponden a la distribución de $u^t_{\vec{x}_0}$ relacionada con el promedio temporal de u, mientras que las gráficas dispuestas en la columna derecha (Gráficas (b), (d) y (f)) muestran la distribución de probabilidad ligada al promedio de ensamble de u[4]. El Apéndice E muestra los histogramas de $u^{t_0}_{\vec{x}}$ obtenidos para determinar si las rejillas de mapeo acoplado aquí examinadas presentan *self-averaging* .

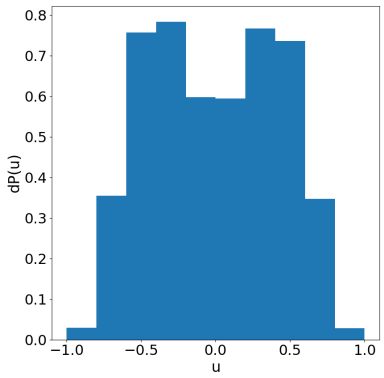
Las gráficas del sistema CML con acoplamiento g_1 (Figuras 4.3, 4.4) para cada semilla muestran gran similitud entre sí, probando así que la rejilla asociada cuenta con ergodicidad. Las discrepancias entre las gráficas son mínimas; su tamaño es propio de ruido en el sistema y sus orígenes más probables son una evolución temporal finita o un número limitado de sistemas para el ensamble. Los cambios provocados por el incremento del periodo transitorio pasan desapercibidos, sugiriendo que un *transient* de 10^3 iteraciones resulta adecuado.

Las rejillas de mapeo acoplado dotadas de acoplamiento g_2 experimentan una situación diferente. Los histogramas elaborados bajo un periodo transitorio de 10^3 iteraciones no coinciden entre sí (Figura 4.5); algunas distribuciones son asimétricas, sin siquiera mostrar una clara preferencia por un lado específico (comparar Gráficas 4.5a y 4.5e). En cambio, las distribuciones obtenidas al contemplar un periodo transitorio más grande sí muestran homología con respecto a los resultados de las simulaciones con g_1 y reflejan ergodicidad (Figura 4.6).

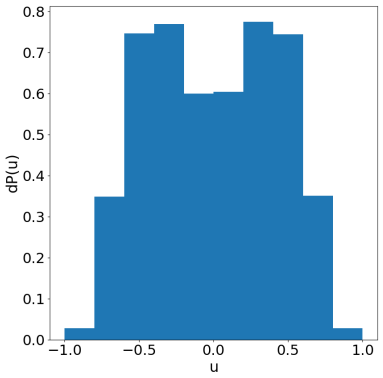
Tomando en cuenta lo anterior, confirmamos que las rejillas de mapeo acoplado definidas en el Capítulo 3 exhiben ergodicidad, al menos para una longitud particular y dos valores de acoplamiento g válidos según [5]. Los resultados aquí planteados indican que la duración del periodo transitorio depende en cierta medida de g y marcan 10^4 iteraciones como tiempo suficiente para el sistema CML con un acoplamiento más cercano al punto crítico $g_c = 0.2054$.

[3]Tal valor para la duración de la evolución temporal fue seleccionado para obtener suficientes resultados en las diferentes simulaciones y pruebas hechas sobre los sistemas CML.

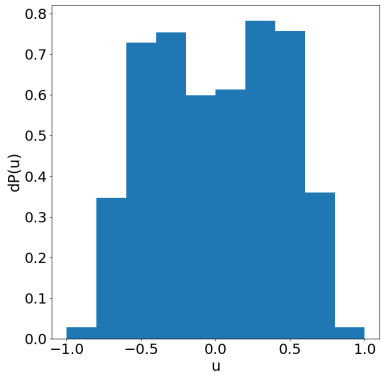
[4]Contemplamos ensambles de ocho sistemas para la construcción de los histogramas pertinentes, al igual que Egolf en [5].



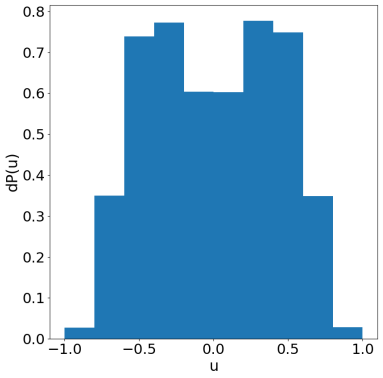
(a) Distribución de $u^t_{\vec{x}_0}$ con un sistema CML, con $\vec{x}_0 = 546$ ($s = 4452967705924205789$)



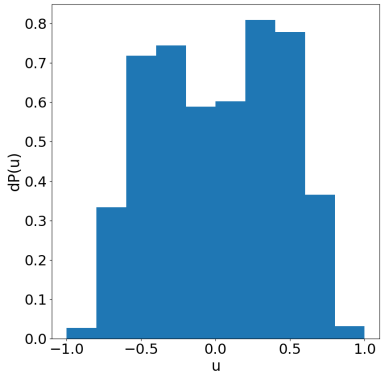
(b) Distribución de $u^t_{\vec{x}_0}$ con un ensamble de 8 sistemas CML ($s = 4452967705924205789$)



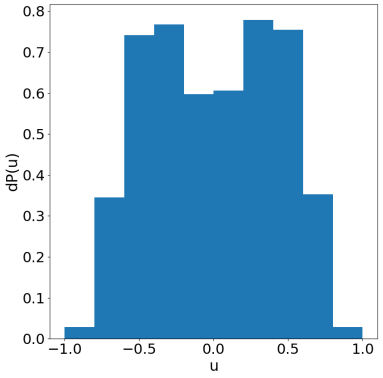
(c) Distribución de $u^t_{\vec{x}_0}$ con un sistema CML, con $\vec{x}_0 = 637$ ($s = 5499070122743521182$)



(d) Distribución de $u^t_{\vec{x}_0}$ con un ensamble de 8 sistemas CML ($s = 5499070122743521182$)

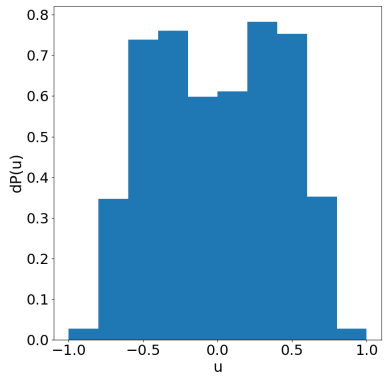


(e) Distribución de $u^t_{\vec{x}_0}$ con un sistema CML, con $\vec{x}_0 = 539$ ($s = 7092030987844978395$)

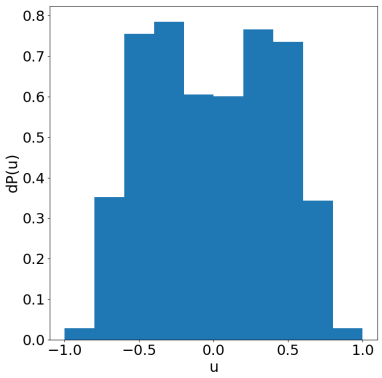


(f) Distribución de $u^t_{\vec{x}_0}$ con un ensamble de 8 sistemas CML ($s = 7092030987844978395$)

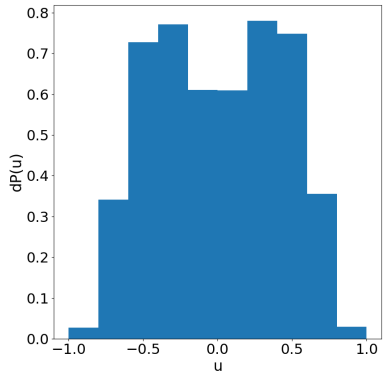
Figura 4.3: Distribuciones de u para rejillas de mapeo acoplado de 32×32 sitios, dotadas de acoplamiento $g = 0.195$ y una evolción temporal de 2×10^5 iteraciones con un periodo transitorio de 10^3 iteraciones, considerando múltiples semillas s



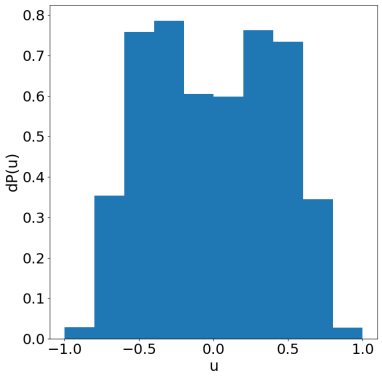
(a) Distribución de $u^t_{\vec{x}_0}$ con un sistema CML, con $\vec{x}_0 = 466$ ($s = 1131207041169938644$)



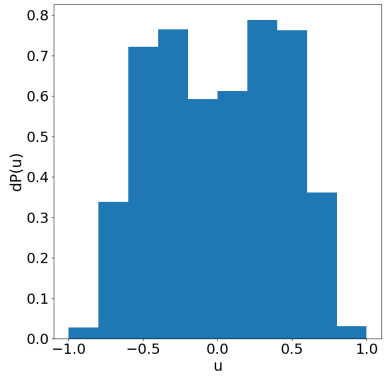
(b) Distribución de $u^t_{\vec{x}_0}$ con un ensamble de 8 sistemas CML ($s = 1131207041169938644$)



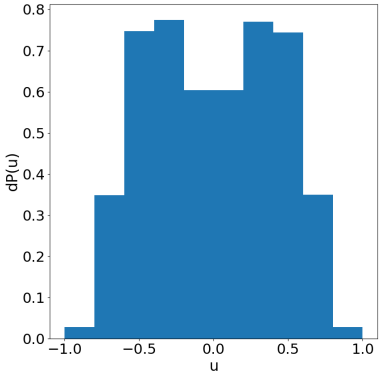
(c) Distribución de $u^t_{\vec{x}_0}$ con un sistema CML, con $\vec{x}_0 = 157$ ($s = 5026122200688730582$)



(d) Distribución de $u^t_{\vec{x}_0}$ con un ensamble de 8 sistemas CML ($s = 5026122200688730582$)

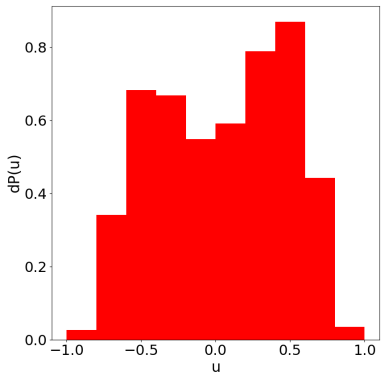


(e) Distribución de $u^t_{\vec{x}_0}$ con un sistema CML, con $\vec{x}_0 = 8$ ($s = 7880188310502227256$)

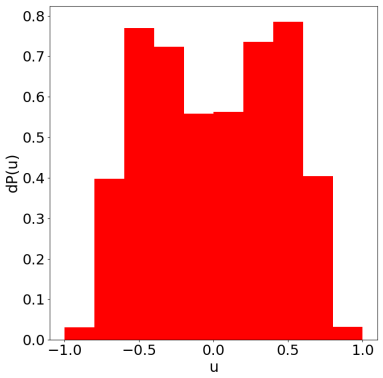


(f) Distribución de $u^t_{\vec{x}_0}$ con un ensamble de 8 sistemas CML ($s = 7880188310502227256$)

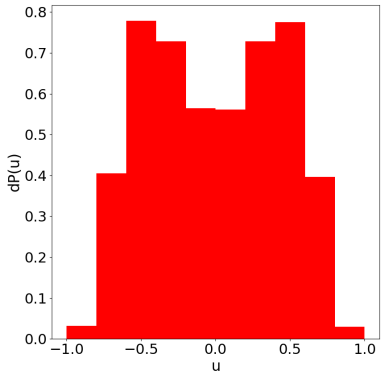
Figura 4.4: Distribuciones de u para rejillas de mapeo acoplado de 32×32 sitios, dotadas de acoplamiento $g = 0.195$ y una evolción temporal de 2×10^5 iteraciones con un periodo transitorio de 10^4 iteraciones, considerando múltiples semillas s



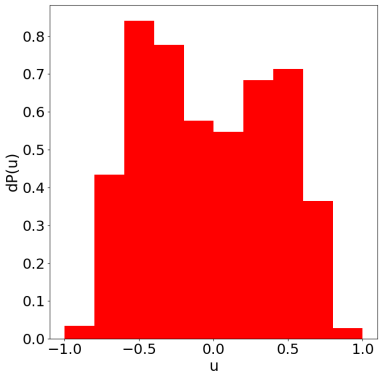
(a) Distribución de $u^t_{\vec{x}_0}$ con un sistema CML, con $\vec{x}_0 = 439$ ($s = 4536755601090925278$)



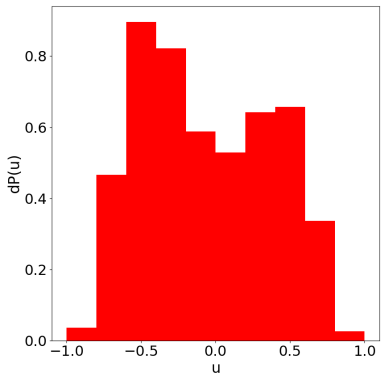
(b) Distribución de $u^t_{\vec{x}_0}$ con un ensamble de 8 sistemas CML ($s = 4536755601090925278$)



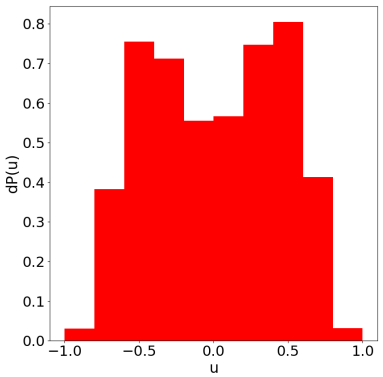
(c) Distribución de $u^t_{\vec{x}_0}$ con un sistema CML, con $\vec{x}_0 = 410$ ($s = 55357595909422011662$)



(d) Distribución de $u^t_{\vec{x}_0}$ con un ensamble de 8 sistemas CML ($s = 55357595909422011662$)

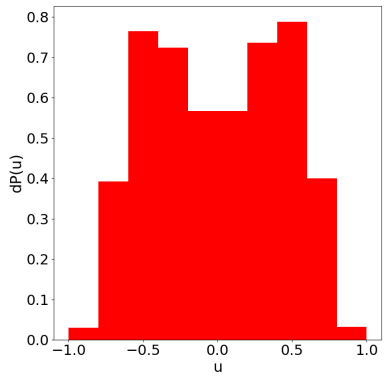


(e) Distribución de $u^t_{\vec{x}_0}$ con un sistema CML, con $\vec{x}_0 = 333$ ($s = 9202787456869569904$)

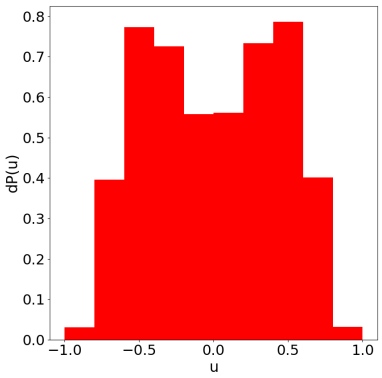


(f) Distribución de $u^t_{\vec{x}_0}$ con un ensamble de 8 sistemas CML ($s = 9202787456869569904$)

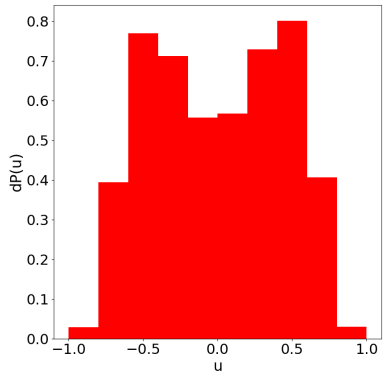
Figura 4.5: Distribuciones de u para rejillas de mapeo acoplado de 32×32 sitios, dotadas de acoplamiento $g = 0.204$ y una evolción temporal de 2×10^5 iteraciones con un periodo transitorio de 10^3 iteraciones, considerando múltiples semillas s



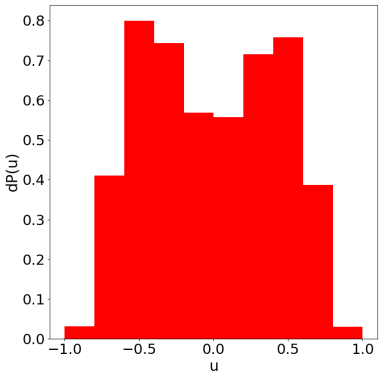
(a) Distribución de $u^t_{\vec{x}_0}$ con un sistema CML, con $\vec{x}_0 = 257$ ($s = 3340065878112466252$)



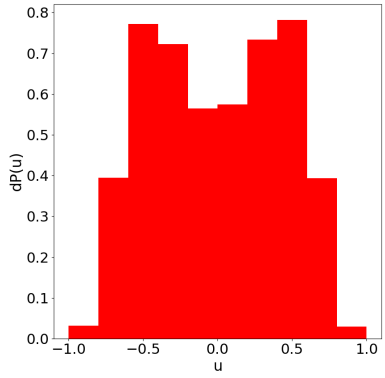
(b) Distribución de $u^t_{\vec{x}_0}$ con un ensamble de 8 sistemas CML ($s = 3340065878112466252$)



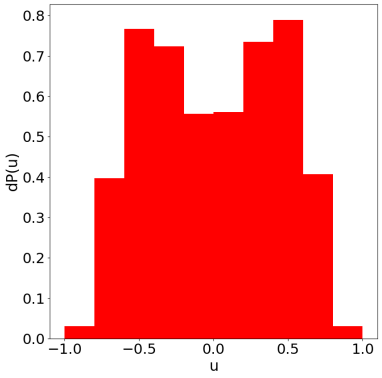
(c) Distribución de $u^t_{\vec{x}_0}$ con un sistema CML, con $\vec{x}_0 = 491$ ($s = 58050196599078125 61$)



(d) Distribución de $u^t_{\vec{x}_0}$ con un ensamble de 8 sistemas CML ($s = 58050196599078125 61$)



(e) Distribución de $u^t_{\vec{x}_0}$ con un sistema CML, con $\vec{x}_0 = 333$ ($s = 6378689186825912380$)



(f) Distribución de $u^t_{\vec{x}_0}$ con un ensamble de 8 sistemas CML ($s = 6378689186825912380$)

Figura 4.6: Distribuciones de u para rejillas de mapeo acoplado de 32×32 sitios, dotadas de acoplamiento $g = 0.204$ y una evolción temporal de 2×10^5 iteraciones con un periodo transitorio de 10^4 iteraciones, considerando múltiples semillas s

En marcado contraste con ergodicidad, la presencia de *self-averaging* es nula, lo cual implicaría que una longitud mayor a $L = 64$ es necesaria para observar dicha propiedad en las

rejillas de mapeo acoplado estudiadas en este trabajo. Cabe recalcar que las distribuciones incluidas en el Apéndice E mantienen cierta relación con el acoplamiento entre sitios de sus respectivas rejillas; los histogramas de los sistemas dotados de acoplamiento g_1 cuentan con dos máximos locales, pero las distribuciones correspondientes a las rejillas de mapeo acoplado con acoplamiento g_2 sólo cuentan con uno.

4.3. Momentos de las desviaciones d

El programa presentado en el Apéndice F calcula varios momentos centrales de promedios $\langle |u| \rangle_G$ para múltiples tamaños de grano G.

Las simulaciones que efectuamos en el estudio de los momentos de las desviaciones $(\langle |u| \rangle_G - \langle |u| \rangle_{x,t})$ fueron sobre sistemas CML con longitudes diferentes y evoluciones temporales de distinta duración.

4.3.1. Detalles Técnicos Computacionales sobre la prueba de comportamiento Gaussiano de los momentos de las desviaciones d

La primera acción del código es pedir al usuario los siguientes parámetros (127-144):

- Semilla para números aleatorios
- Longitud L de la rejilla de mapeo acoplado
- Acoplamiento g entre los sitios del sistema CML
- Duración total y duración del período de transición (*transient*) de la evolución temporal de la rejilla
- Número total y valores de los tamaños de grano G
- Número total y valores de los momentos a calcular [5]

Con tales datos, el generador de números aleatorios recibe una semilla (146-148) y el programa construye la gráfica de la rejilla y su correspondiente lista de primeros vecinos (149-154). Dicha gráfica evoluciona posteriormente de acuerdo a la regla de actualización mencionada en las Secciones 1 y 3 (157-157).

Luego, el programa calcula los momentos centrales de los promedios de $|u|$ sobre "subrejillas" de $G \times G$ sitios (159-167), como queda ilustrado en la Figura 4.7. Dado un tamaño de grano G, la función `calc_uG` (108-122) estima los promedios $\langle |u| \rangle_G$ de todas las "subrejillas" en el sistema y genera un arreglo con ellos (176). Después, los momentos de las desviaciones entre los elementos de dicho arreglo y su promedio son calculados implementando la función `moment` de `SciPy`, la cual calcula los momentos centrales con respecto al promedio de los elementos en su entrada (165-166). Una vez completado el procedimiento anterior para todos los tamaños de

[5] El usuario cuenta con la posibilidad de calcular y guardar momentos adicionales a los que Egolf analiza en [5]. Sin embargo, el programa únicamente puede graficar los momentos reportados en el artículo con la debida normalización.

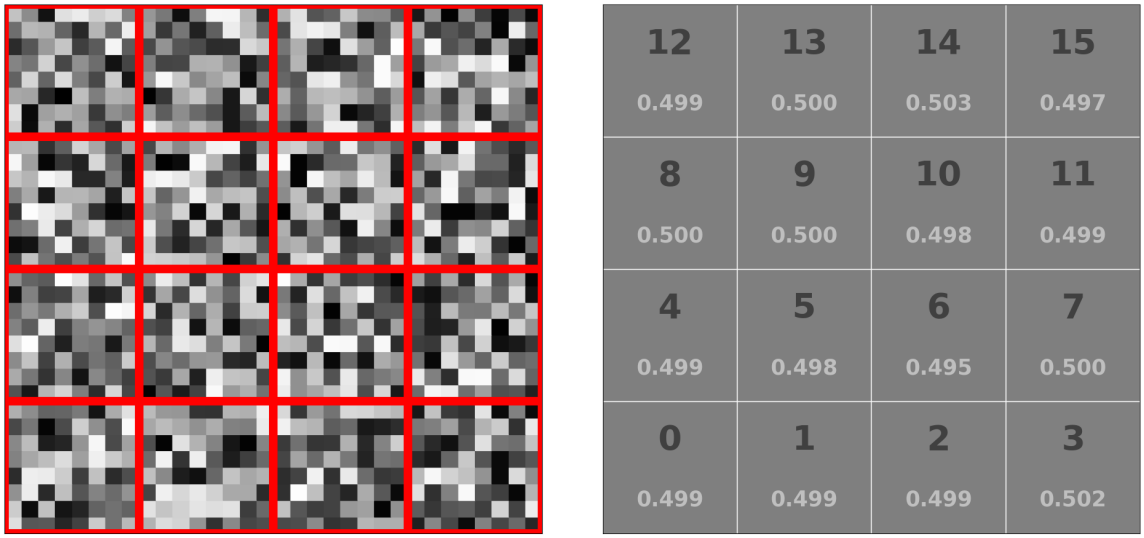


Figura 4.7: Esquema de un sistema CML de 32×32 sitios (izq.) y las diferentes "subrejillas" en él dado un grano de longitud $G = 8$ (der.). La escala de grises en la rejilla de mapeo acoplado marca el valor de u en cada sitio, siendo negro y blanco los colores asociados a -1 y $+1$ respectivamente. En cambio, en la representación de las "subrejillas" en tal sistema, la escala de grises denota las magnitudes de los promedios $\langle |u| \rangle_G$, calculados sobre todos los sitios dentro del cuadrado con bordes rojos asociado a cada "subrejilla". Dichos valores oscilan entre 0 (negro) y 1 (blanco). Las diferencias entre los tonos de cada región son ínfimas, por lo que el valor de cada promedio queda escrito con un color gris claro debajo del número de su correspondiente "subrejilla".

grano G, los momentos son almacenados en archivos de texto (168-170) y graficados con ciertos factores de escala en una representación logarítmica (172-203)[6].

4.3.2. Resultados de la prueba de comportamiento Gaussiano de los momentos de las desviaciones d

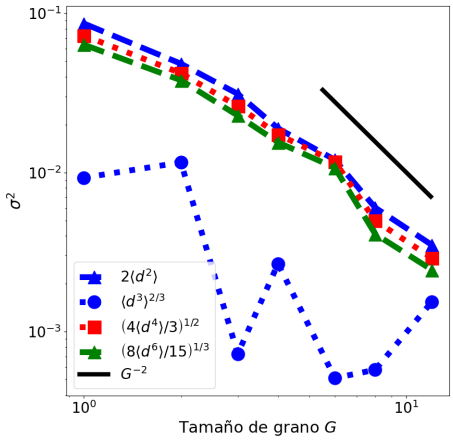
Comenzamos el análisis de los momentos de las desviaciones $d = \langle |u| \rangle_G - \langle |u| \rangle_{x,t}$ con un sistema CML de 24×24 sitios, acoplamiento $g = 0.195$ y granos de magnitud $G = [1, 2, 3, 4, 6, 8, 12]$. Consideramos tres duraciones distintas para la evolución temporal del sistema: 2×10^5, 5×10^5 y 10^6 iteraciones. La Figura 4.8 muestra los resultados de las simulaciones de tal rejilla. Las tres gráficas en tal figura indican que los momentos pares (con los debidos factores de escala) decrecen con G de la misma forma y son todos mayores que el tercer momento, independientemente del tamaño de grano G o la duración de su evolución temporal. Si bien estas características apuntan a un comportamiento Gaussiano, existen ciertos detalles que imposibilitan afirmar tal hecho definitivamente. El más significativo de ellos consiste en que el decrecimiento del tercer

[6]Algunos de los valores para el tercer momento de las desviaciones d fueron negativos y usar el factor de escala que Egolf en ellos producía valores indefinidos. Como la función $f(x) = x^{2/3}$ es par, opté por considerar su valor absoluto al construir las gráficas

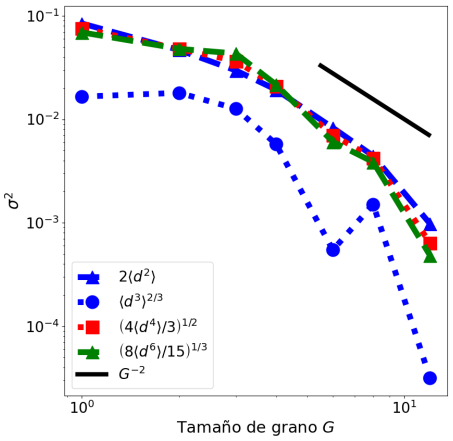
momento no es tan dramático como lo sugiere Egolf[7]; de hecho, la brecha entre los momentos pares y éste alcanza un mínimo para $G = 12$ (salvo en la Figura 4.8b). Además, los momentos pares de las simulaciones con sistemas con evoluciones temporales de 5×10^5 y 10^6 iteraciones experimentan una caída abrupta para los granos más grandes que no coincide con el cambio ideal para una variable aleatoria normal; los momentos pares de la simulación con una evolución temporal de 2×10^5 iteraciones son la excepción, puesto que sí decrecen como G^{-2} (con ligeras fluctuaciones).

Ejecutamos otras simulaciones similares sobre rejillas de 32×32 sitios (4.9) y 48×48 sitios (4.10), con los mismos tres intervalos para la evolución temporal. En este caso, el tercer momento exhibe el comportamiento esperado, especialmente en las Gráficas 4.9b y 4.9c, y los momentos pares en ambos sistemas parecen comportarse como ruido blanco; el decrecimiento drástico asociado a los tamaños de grano más grandes en las simulaciones sobre el sistema de 24×24 sitios no aparece en gráfica alguna y las discrepancias entre las mediciones experimentales y la recta (en la gráfica con escala logarítmica) que decae como G^{-2} es mínimo. Posiblemente, la rejilla con longitud $L = 24$ no era suficientemente grande.

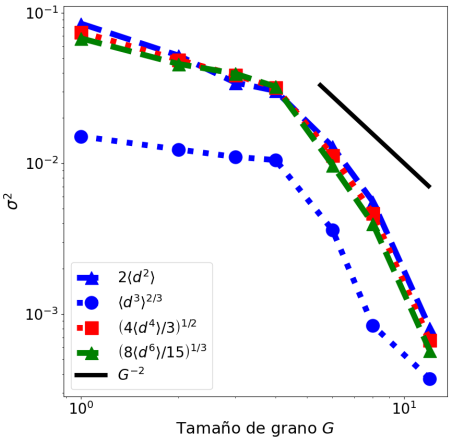
[7]La ausencia de los valores de $\langle d \rangle^{\frac{2}{3}}$ en la Figura 1.B en [5] impide hacer una comparación más precisa con los resultados aquí presentados.



(a) Momentos $\langle d\rangle^n$ en sistema CML tras 2×10^5 iteraciones

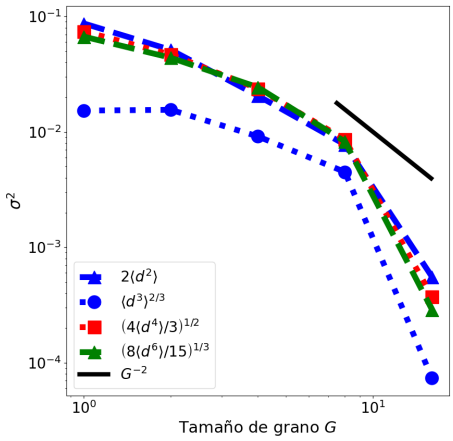


(b) Momentos $\langle d\rangle^n$ en sistema CML tras 5×10^5 iteraciones

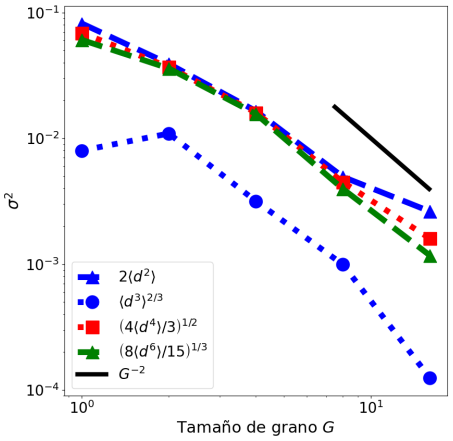


(c) Momentos $\langle d\rangle^n$ en sistema CML tras 10^6 iteraciones

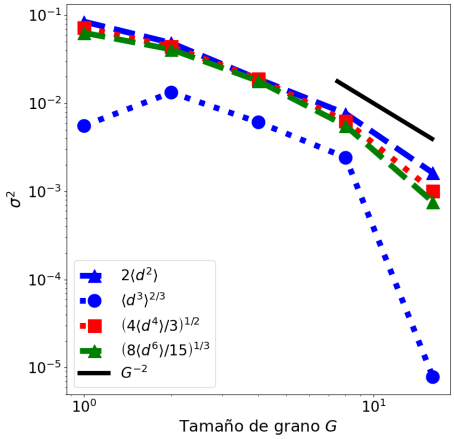
Figura 4.8: Momentos de las desviaciones $d = (\langle|u|\rangle_G - \langle|u|\rangle_{x,t})$ (con factores de escala usados por Egolf) para una rejilla de mapeo acoplado de 24×24 sitios, con un acoplamiento $g = 0.195$ y granos de tamaño $G = [1, 2, 3, 4, 6, 8, 12]$.



(a) Momentos $\langle d\rangle^n$ en sistema CML tras 2×10^5 iteraciones

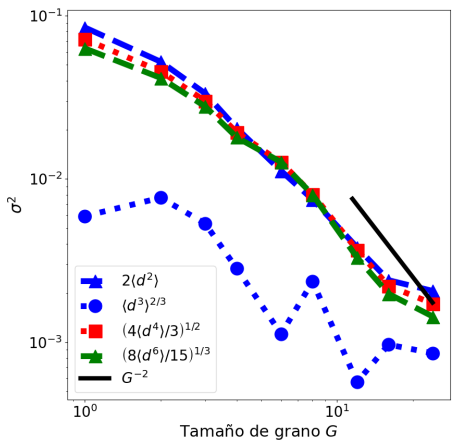


(b) Momentos $\langle d\rangle^n$ en sistema CML tras 5×10^5 iteraciones

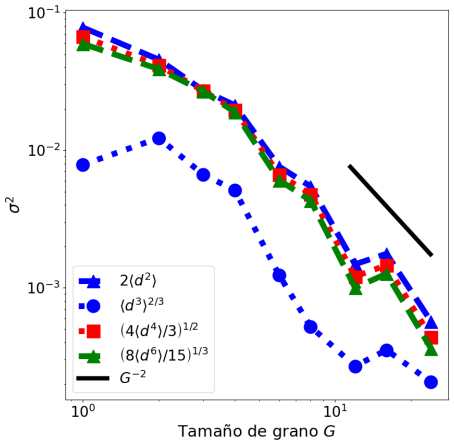


(c) Momentos $\langle d\rangle^n$ en sistema CML tras 10^6 iteraciones

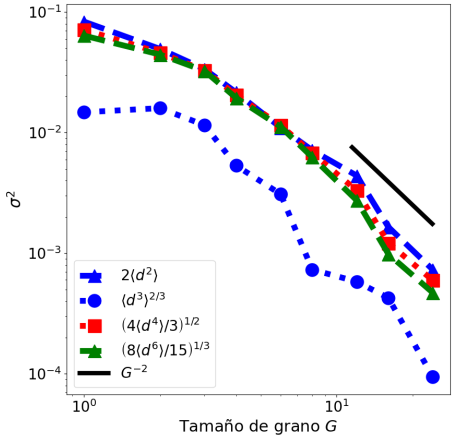
Figura 4.9: Momentos de las desviaciones $d = (\langle|u|\rangle_G - \langle|u|\rangle_{x,t})$ (con factores de escala usados por Egolf) para una rejilla de mapeo acoplado de 32×32 sitios, con un acoplamiento $g = 0.195$ y granos de tamaño $G = [1, 2, 4, 8, 16]$.



(a) Momentos $\langle d\rangle^n$ en sistema CML tras 2×10^5 iteraciones



(b) Momentos $\langle d\rangle^n$ en sistema CML tras 5×10^5 iteraciones



(c) Momentos $\langle d\rangle^n$ en sistema CML tras 10^6 iteraciones

Figura 4.10: Momentos de las desviaciones $d = (\langle|u|\rangle_G - \langle|u|\rangle_{x,t})$ (con factores de escala usados por Egolf) para una rejilla de mapeo acoplado de 48×48 sitios, con un acoplamiento $g = 0.195$ y granos de tamaño $G = [1, 2, 3, 4, 6, 8, 12, 16, 24]$.

4.4. Balance Detallado

El código en el Apéndice H define una rejilla de mapeo acoplado a grano grueso y replica el procedimiento usado por Egolf para determinar la presencia de balance detallado en tal sistema.

Los sistemas CG-CML que estudiamos con este código cuentan todos con las mismas dimensiones. Sin embargo, las rejillas de mapeo acoplado "finas" usadas para definirlos difieren en su longitud y el tamaño de grano correspondiente.

4.4.1. Detalles Técnicos Computacionales sobre la prueba de Balance Detallado

El programa comienza solicitando los siguientes datos al usuario (130-139):

- Longitud L y acoplamiento g de la rejilla de mapeo acoplado a usar como base en la construcción de la rejilla "a grano grueso"
- Tamaño de grano G
- Duración total y duración del período transitorio de la evolución temporal del sistema CML de $L \times L$ sitios
- Duración total de la evolución temporal del sistema CG-CML, junto con intervalo ΔT usado para calcular promedios temporales

Una vez inicializado el generador de números aleatorios (con una semilla seleccionada siguiendo el mismo proceso que en las simulaciones anteriores) y suministrados valores para los parámetros enlistados anteriormente (131-143), el código construye un sistema CML y facilita su evolución temporal (144-155).

Luego, la simulación genera y actualiza un sistema CG-CML a partir del sistema CML previamente definido (157-223). Tras la iniciación de varios arreglos (158-164) y los nombres de los archivos destinados a guardar datos relevantes (166-168), comienza un ciclo `for` que produce la evolución temporal de la rejilla a grano grueso (172-218). Dentro de este *loop*, hay otro ciclo `for` de ΔT iteraciones (177-205) que calcula los promedios espaciales de la variable u sobre las $\left(\frac{L}{G}\right)^2$ distintas "subrejillas" de $G \times G$ sitios disponibles en la rejilla de $L \times L$ sitios (178-186), la actualización de tal rejilla conforme a la regla planteada en la Ecuación 1.1 (188-198) y el almacenamiento de la configuración del sistema CML en un archivo CSV (199-204)[8]. Posteriormente, el código implementa la función `sign_modif` (113-126) para estimar los promedios temporales del conjunto de variables $\langle u^T_{\vec{y}} \rangle_{CG(\vec{x})}$ recopiladas durante las ΔT iteraciones del sistema CML y asociar a cada uno de tales promedios una variable $\tilde{v}^T_{\vec{x}}$ (205-207), definida como

$$\tilde{v}^T_{\vec{x}} = \begin{cases} 1 & \tilde{u}^T_{\vec{x}} = +1 \\ 0 & \tilde{u}^T_{\vec{x}} = -1 \end{cases} \tag{4.1}$$

siendo $\tilde{u}^T_{\vec{x}}$ el signo del promedio temporal y espacial (sobre la "subrejilla" $CG(\vec{x})$) de u (Ecuación 3.3). En caso de que $\langle u^T_{\vec{y}} \rangle_{CG(\vec{x}),\Delta T}$ fuera nulo al calcularlo numéricamente, el valor de $\tilde{v}^T_{\vec{x}}$ alternaría entre 1 y 0 (120-126).

Si bien las definiciones de $\tilde{v}^T_{\vec{x}}$ y $\tilde{u}^T_{\vec{x}}$ no coinciden perfectamente, ambas variables portan la misma información sobre la rejilla "a grano grueso". El uso de $\tilde{v}^T_{\vec{x}}$ es preferente por la facilidad que proporciona en la indexación de los estados disponibles al sistema CG-CML (208-215); al asociar a cada "subrejilla" con posición $\vec{x}$ un entero $n(\vec{x})$, la identidad de cada estado S_i disponible al sistema queda establecida como

[8]La configuración de la rejilla de mapeo acoplado "fina" no proporciona información necesaria en alguno de los procesos subsecuentes de la prueba de balance detallado. Guardar tales datos únicamente sirve para verificar el debido funcionamiento del programa y contar con un respaldo de los resultados. Por consiguiente, el almacenamiento de los archivos CSV resultantes, los cuales consumen una vasta cantidad de memoria, es opcional.

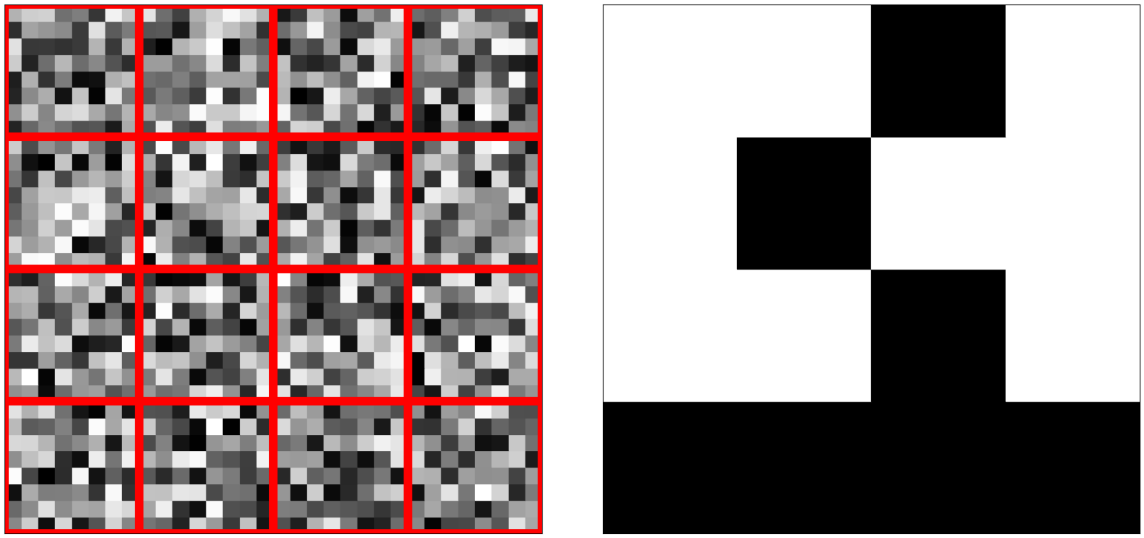


Figura 4.11: Sistema CML de 32×32 sitios con acoplamiento $g = 0.204$ (izq.) y sistema CG-CML asociado con un grano $G = 8$ y un intervalo temporal $\Delta T = 100$ (der.) al tiempo $t = 2 \times 10^5$. La escala de grises representa el valor de u en cada sitio, siendo negro y blanco los colores asociados a -1 y $+1$ respectivamente. Bajo la regla de indexación planteada en la Ecuación 4.2, la rejilla "a grano grueso" aparece dispuesta en el estado $S_i = 48560$.

$$S_i = \sum_{\vec{x}} 2^{n(\vec{x})} \cdot \tilde{v}_{\vec{x}}^T \tag{4.2}$$

La Figura 4.11 ilustra el procedimiento anterior para una rejilla de mapeo acoplado de 32×32 y un grano de longitud $G = 8$.

Cuando el ciclo `for` termina, la evolución completa de la rejilla a grano grueso es almacenada en un archivo de texto (219-220).

Después, el programa realiza un barrido sobre el arreglo que lleva la historia del sistema CG-CML y cuenta las transiciones $T_{i\to j}$ y $T_{j\to i}$ entre cada par de estados (i, j) disponibles a la rejilla a grano grueso (224-229). Una vez concluido tal proceso, el código calcula las diferencias $\Delta^{(i,j)} = T_{i\to j} - T_{j\to i}$ y el total de transiciones $N^{(i,j)} = T_{i\to j} + T_{j\to i}$ de cada par de estados (i, j) (siendo $i < j$) (233-243) y guarda (gradualmente) la suma de Δ^2 y el número de pares de estados (i, j) correspondientes a cada valor N disponible a $N^{(i,j)}$ en un diccionario, usando N como clave (244-249). El diccionario resultante es posteriormente almacenado como un archivo de texto (251-252). Finalmente, los promedios $\left\langle \left(\frac{\Delta}{N}\right)^2 \right\rangle$ son derivados de las entradas de dicho diccionario (256-269), la suma χ^2 de los cuadrados de las diferencias entre tales promedios "empíricos" y los valores predichos por la Ecuación 3.5 es calculada (271-276) y los resultados obtenidos quedan ilustrados en una gráfica (278-299).

4.4.2. Resultados de la prueba de Balance Detallado

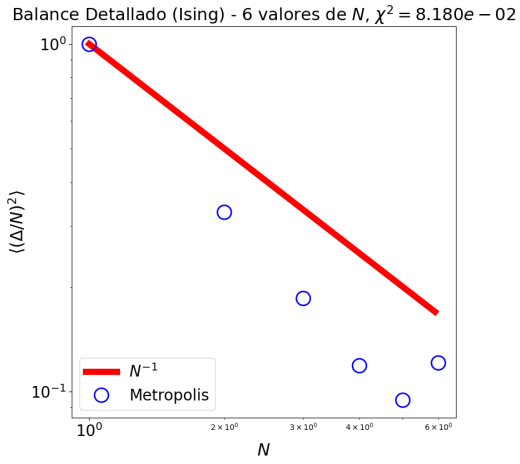
Antes de analizar el sistema de Egolf, evaluamos la condición de balance detallado en el modelo de Ising usando el código que aparece en las líneas 1-123 y 245-367 en el Apéndice K, que es prácticamente idéntico al programa recién descrito. La única diferencia entre las dos versiones es el uso del algoritmo de Metropolis para actualizar el sistema en lugar de la regla en la Ecuación 3.1 y los promedios temporales sobre subrejillas de $G \times G$ sitios antes descritos. Los resultados obtenidos para sistemas de Ising con longitud $L = 4$ y acoplamiento $J = 0.05$ aparecen en la Figura 4.12, con tiempos de evolución de 1.6×10^5, 1.6×10^6 y 1.6×10^7 iteraciones[9].

Las distribuciones de los promedios experimentales en las Gráficas 4.12b y 4.12c sugieren a simple vista la presencia de balance detallado en la rejilla de Ising. Sin embargo, las variaciones de χ^2 con respecto al tiempo de evolución imposibilitan hacer una afirmación concluyente. La dispersión de los promedios respecto a su comportamiento teórico (dada por la magnitud de χ^2) debería disminuir al aumentar la duración de la evolución temporal del sistema, ya que las tasas de transición entre estados y sus correspondientes probabilidades son mejores aproximaciones a los valores "reales" (asociados a una evolución temporal infinita) al ser calculadas bajo tiempos de evolución más largos. Contrario a esto, χ^2 incrementa a 5.159×10^{-2} al considerar una evolución de 10^6 iteraciones. A pesar de ello, argumentamos que tal hecho no refuta que el sistema presente balance detallado, sino que es producto de la finitud de la simulación y el mayor número de valores distintos para el total de transiciones N; en las tres gráficas, las diferencias entre los puntos empíricos y "teóricos" son más apreciables para los valores más grandes de N en las tres gráficas de la Figura 4.12 y el total de valores diferentes para N se vuelve naturalmente mayor conforme crece el tiempo de evolución, lo cual implica a su vez la inclusión de más términos en la suma que define χ^2. Además, la simulación con 10^6 iteraciones no supera el máximo valor de χ^2, asociado al sistema con una evolución de 1.6×10^5 iteraciones. Tomando en cuenta tales observaciones, juzgamos que las Gráficas4.12b y 4.12c prueban la presencia de balance detallado en los sistemas de Ising examinados y así verifican el correcto funcionamiento del código.

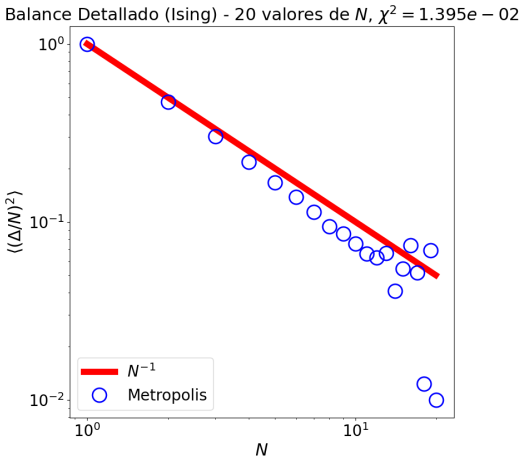
Realizamos entonces simulaciones sobre sistemas CG-CML de 4×4 sitios a partir de rejillas de mapeo acoplado con acoplamiento $g = 0.204$. Consideramos rejillas "finas" con longitudes y tamaños de grano $\{L = 8, G = 2\}$ y $\{L = 32, G = 8\}$ con una evolución temporal de 2×10^5 iteraciones y un periodo transitorio de 10^4 iteraciones. Contemplamos tiempos de evolución para ambas clases de rejillas "a grano grueso" de 2×10^5 y 4×10^5 iteraciones. Los resultados aparecen en las Figuras 4.13 y 4.14.

Las simulaciones para los dos tipos de sistemas CG-CML revelan un decrecimiento del promedio de $\left(\Delta^{(i,j)}/N^{(i,j)}\right)^2$ aproximadamente igual a N^{-1}, aunque la dispersión de los puntos experimentales asociados a las rejillas "a grano grueso" derivada de sistemas CML con longitud $L = 8$ supera significativamente la de los valores empíricos correspondientes a los otros sistemas CG-CML; χ^2 en las Gráficas 4.13a y 4.13b es un orden de magnitud mayor que en las Gráficas 4.14a y 4.14b, incluso con cantidades similares de valores distintos del total de transiciones N, en constraste con los resultados de la simulación sobre rejillas de Ising (Figura 4.12). La causa más probable detrás de esto es la predominancia de promedios experimentales con magnitudes superiores a sus respectivos valores teóricos en los resultados asociados a sistemas CG-CML

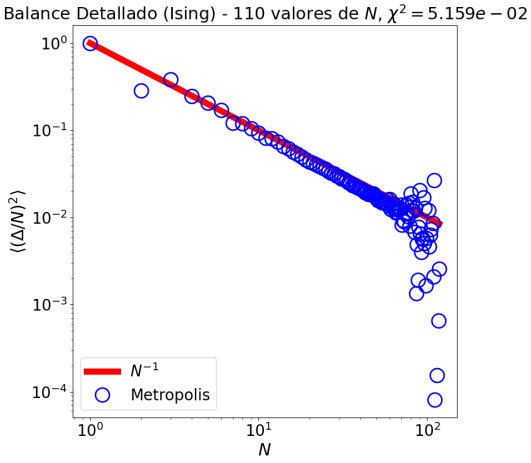
[9]El usuario proporciona al inicio de la simulación para las rejillas de Ising el número (promedio) de veces o rondas que será aplicado el algoritmo de Metropolis por cada sitio en el sistema. Por ende, una rejilla cuadrada de 16 sitios en una simulación de 10^4 rondas experimentará una evolución temporal (regida por Metropolis) de 1.6×10^5



(a) $\left\langle \left(\Delta^{(i,j)}/N^{(i,j)}\right)^2 \right\rangle$ en rejilla de Ising de 4×4 sitios con una evolución temporal de 1.6×10^5 iteraciones y semilla $s = 6229836497689036069$



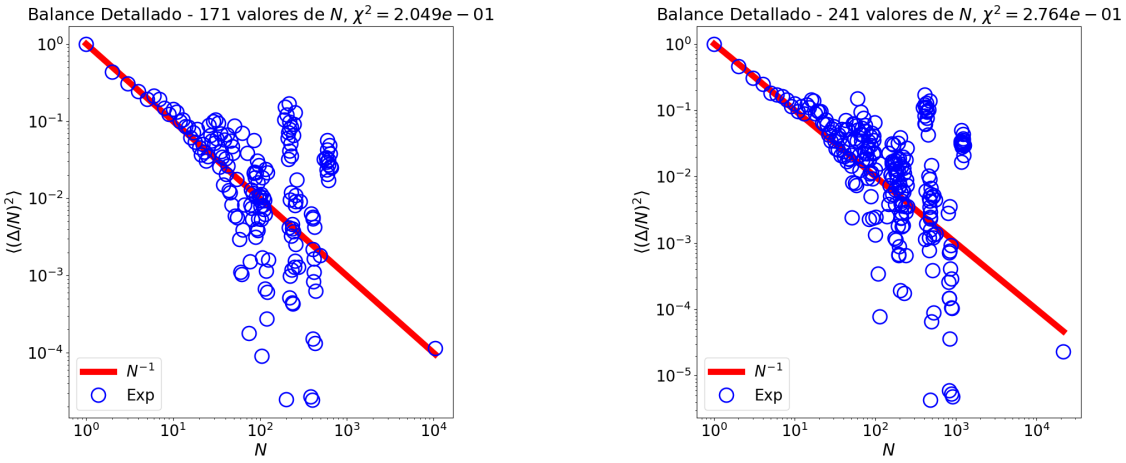
(b) $\left\langle \left(\Delta^{(i,j)}/N^{(i,j)}\right)^2 \right\rangle$ en rejilla de Ising de 4×4 sitios con una evolución temporal de 1.6×10^6 iteraciones y semilla $s = 1016131477794588868$



(c) $\left\langle \left(\Delta^{(i,j)}/N^{(i,j)}\right)^2 \right\rangle$ en rejilla de Ising de 4×4 sitios con una evolución temporal de 1.6×10^7 iteraciones y semilla $s = 7450712230650374706$

Figura 4.12: Comportamiento de $\left\langle \left(\Delta^{(i,j)}/N^{(i,j)}\right)^2 \right\rangle$ con respecto al total de transiciones entre pares de estados (i, j) disponibles a sistemas de Ising de 4×4 sitios, con acoplamiento $J = 0.030$. Los círculos azules marcan los valores del promedio $\left\langle \left(\Delta^{(i,j)}/N^{(i,j)}\right)^2 \right\rangle$ para las diferentes mediciones de $\Delta^{(i,j)}$ dado un total de transiciones $N^{(i,j)} = N$, mientras que la línea roja representa N^{-1}, el comportamiento esperado de un sistema con balance detallado.

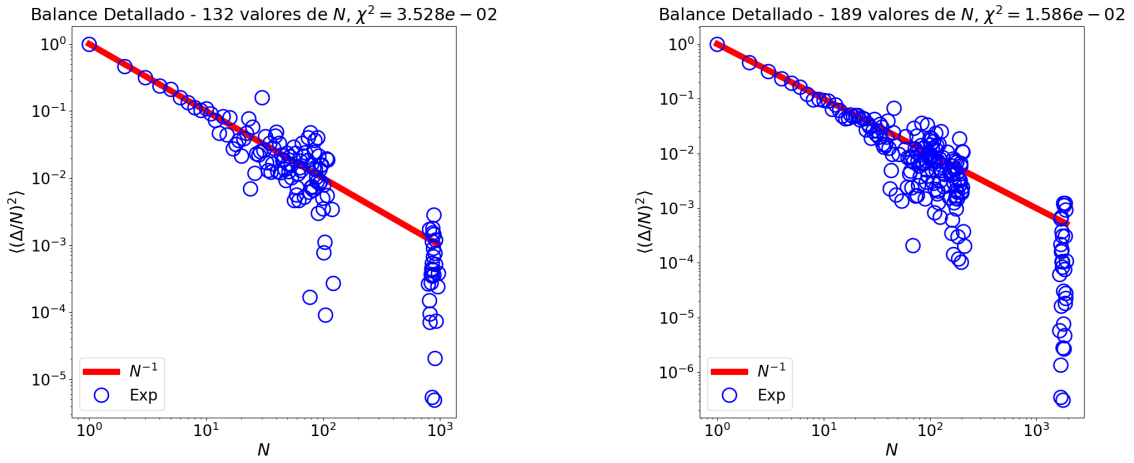
construidos sobre rejillas de mapeo acoplado con $L = 8$ y $G = 2$, apreciables en sus gráficas como círculos azules por encima de la línea roja. Al usar una escala logarítmica en el eje vertical, las diferencias entre dichos puntos y el comportamiento teórico esperado (Ecuación 3.5) son una contribución más grande en el cálculo de χ^2 que aquellas correspondientes a puntos que aparecen debajo de la recta inclinada. Si bien hay discrepancias visibles entre los resultados empíricos y los valores predichos para N grande en las gráficas de la Figura 4.14, éstas son muy pequeñas en relación a las que aparecen en las de la Figura 4.13 al ser generadas por puntos menores que N^{-1}. Además, dichas ínfimas diferencias podrían ser consecuencia de una evolución temporal finita "corta" e incrementar el tiempo de evolución podría aumentar los valores de N al grado suficiente para eliminarlas (o al menos reducirlas). Por lo tanto, afirmarmos que los sistemas CG-CML relacionados con rejillas "finas" de longitud $L = 32$ y grano $G = 8$, en contraste con las rejillas "a grano grueso" asociadas a sistemas CML con $L = 8$ y $G = 2$, sí presentan balance detallado, conforme a los límites planteados en [5].



(a) $\left\langle \left(\Delta^{(i,j)}/N^{(i,j)}\right)^2 \right\rangle$ en sistema CG-CML de 4×4 sitios con una evolución temporal de 2×10^5 iteraciones y semilla $s = 3249191637585704358$

(b) $\left\langle \left(\Delta^{(i,j)}/N^{(i,j)}\right)^2 \right\rangle$ en sistema CG-CML de 4×4 sitios con una evolución temporal de 4×10^5 iteraciones y semilla $s = 5617058624460882590$

Figura 4.13: Comportamiento de $\left\langle \left(\Delta^{(i,j)}/N^{(i,j)}\right)^2 \right\rangle$ con respecto al total de transiciones entre pares de estados (i, j) disponibles a sistemas CG-CML de 4×4 sitios, elaborados a partir de rejillas de mapeo acoplado de longitud $L = 8$, tamaño de grano $G = 2$ y acoplamiento $g = 0.204$. Los círculos azules marcan los valores del promedio $\left\langle \left(\Delta^{(i,j)}/N^{(i,j)}\right)^2 \right\rangle$ para las diferentes mediciones de $\Delta^{(i,j)}$ dado un total de transiciones $N^{(i,j)} = N$, mientras que la línea roja representa N^{-1}, el comportamiento esperado de un sistema con balance detallado. Se consideró $\Delta T = 100$ para el cálculo de los promedios temporales.



(a) $\left\langle \left(\Delta^{(i,j)}/N^{(i,j)}\right)^2 \right\rangle$ en sistema CG-CML de 4×4 sitios con una evolución temporal de 2×10^5 iteraciones y semilla $s = 316581566959966622$

(b) $\left\langle \left(\Delta^{(i,j)}/N^{(i,j)}\right)^2 \right\rangle$ en sistema CG-CML de 4×4 sitios con una evolución temporal de 4×10^5 iteraciones y semilla $s = 3759620218372826529$

Figura 4.14: Comportamiento de $\left\langle \left(\Delta^{(i,j)}/N^{(i,j)}\right)^2 \right\rangle$ con respecto al total de transiciones entre pares de estados (i, j) disponibles a sistemas CG-CML de 4×4 sitios, elaborados a partir de rejillas de mapeo acoplado de longitud $L = 32$, tamaño de grano $G = 8$ y acoplamiento $g = 0.204$. Los círculos azules marcan los valores del promedio $\left\langle \left(\Delta^{(i,j)}/N^{(i,j)}\right)^2 \right\rangle$ para las diferentes mediciones de $\Delta^{(i,j)}$ dado un total de transiciones $N^{(i,j)} = N$, mientras que la línea roja representa N^{-1}, el comportamiento esperado de un sistema con balance detallado. Se consideró $\Delta T = 100$ para el cálculo de los promedios temporales.

Nos sorprendió inicialmente observar gráficas diferentes a las que aparecen en [5]. Tras revisar minuciosamente el artículo una vez más, notamos que los promedios de $\left(\Delta^{(i,j)}/N^{(i,j)}\right)^2$ fueron calculados tomando en cuenta los factores $\Delta^{(i,j)}$ ligados a valores de N agrupados en *bins* cuyo ancho incrementa exponencialmente con el total de transiciones[10]. El texto no proporciona una razón específica detrás de esta decisión. Suponemos que fue un intento por reducir el efecto de una evolución temporal finita sobre la dispersión de los promedios respecto a su comportaimento teórico.

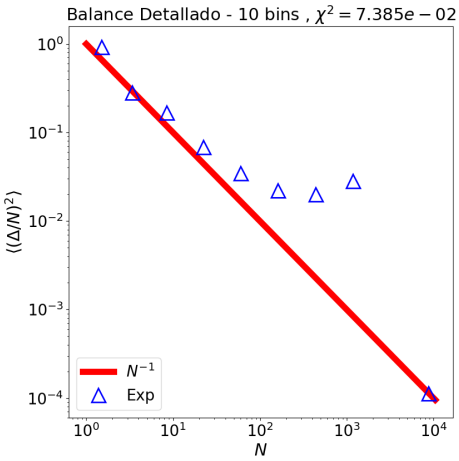
Graficando los resultados previos bajo tales consideraciones con el código en el Apéndice I[11], obtuvimos las gráficas presentadas en las Figuras 4.15 y 4.16.

Los resultados conseguidos contradicen nuestra conjetura sobre la explicación detrás del *binning* de las mediciones de $\Delta^{(i,j)}$ para calcular sus promedios, aunque parezca superficialmente

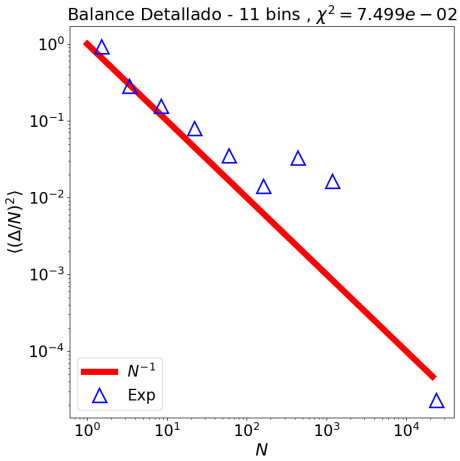
[10] Estas especificaciones son mencionadas al pie de la Figura 3.A en [5].

[11] El correcto uso del código requiere almacenar archivos en carpetas siguiendo una nomenclatura específica. Por ejemplo, el código en el Apéndice I buscaría uno de los archivos generados por el código para una prueba de balance detallado (Apéndice H) efectuada sobre un sistema CML de 32×32 sitios, con un tamaño de grano $G = 8$, acoplamiento $g = 0.204$, evolución temporal (para rejilla "fina" y "a grano grueso") de 2×10^5 iteraciones, un periodo de transición de 10^4 iteraciones y semilla 2021 en la carpeta `Res_BD_L32_G8_g0.204_Niter2.000e+05_trans1.000e+04_Nrondas2.000e+05_seed2021`.

que la confirman. De acuerdo a la discusión anterior sobre el efecto de promedios numéricos superiores a sus valores teóricos, el primer punto en las Gráficas 4.16a y 4.16b es suficientemente grande para exceder la suma de las diferencias entre puntos experimentales y la recta N^{-1} en las Gráficas 4.15a y 4.15b. A pesar de la notoria separación entre experimento y teoría que muestran las gráficas en la Figura 4.15, la similitud de la dispersión de los promedios calculados en la simulación (dado por χ^2) sugeriría entonces que las dos clases de rejillas "a grano grueso" o exhiben ambas balance detallado o carecen de él, en oposición al límite reportado en [5]. Ante la ausencia de un propósito claro para el agrupamiento de los valores $\Delta^{(i,j)}$ en *bins* y el contraste con los resultados ilustrados en las Figuras 4.13 y 4.14, consideramos óptimo juzgar la presencia de balance detallado únicamente a partir de las primeras simulaciones analizadas en esta Sección.

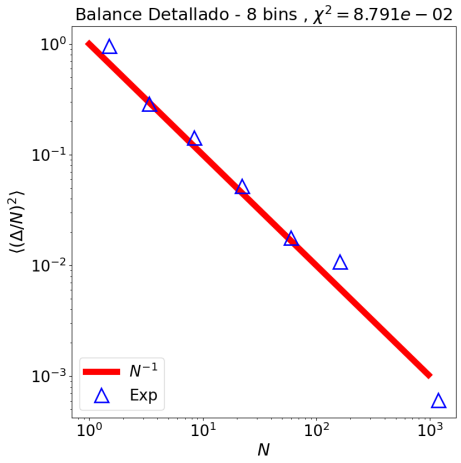


(a) $\left\langle \left(\Delta^{(i,j)}/N^{(i,j)}\right)^2 \right\rangle$ sobre *bins* en sistema CG-CML de 4×4 sitios con una evolución temporal de 2×10^5 iteraciones y semilla $s = 3249191637585704358$

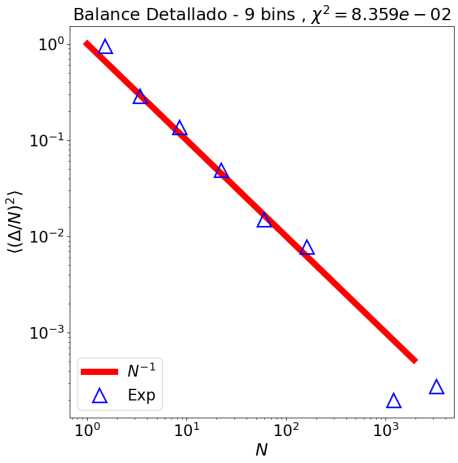


(b) $\left\langle \left(\Delta^{(i,j)}/N^{(i,j)}\right)^2 \right\rangle$ en sistema CG-CML de 4×4 sitios con una evolución temporal de 4×10^5 iteraciones y semilla $s = 5617058624460882590$

Figura 4.15: Dependencia de $\left\langle \left(\Delta^{(i,j)}/N^{(i,j)}\right)^2 \right\rangle$ con el total de transiciones entre pares de estados (i, j) disponibles a sistemas CG-CML de 4×4 sitios, elaborados a partir de rejillas de mapeo acoplado de longitud $L = 8$, tamaño de grano $G = 2$ y acoplamiento $g = 0.204$. Los círculos azules denotan los promedios $\left\langle \left(\Delta^{(i,j)}/N^{(i,j)}\right)^2 \right\rangle$ para las diferentes mediciones de $\Delta^{(i,j)}$ para valores similares del total de transiciones $N^{(i,j)}$ y la línea roja marca la función N^{-1}. Se consideró $\Delta T = 100$ para el cálculo de los promedios temporales. La tasa de crecimiento de la función exponencial usada para definir los *bins* que agrupan mediciones de $\Delta^{(i,j)}$ fue igual a 1.



(a) $\left\langle \left(\Delta^{(i,j)}/N^{(i,j)}\right)^2 \right\rangle$ sobre *bins* en sistema CG-CML de 4×4 sitios con una evolución temporal de 2×10^5 iteraciones y semilla $s = 3316581566959966622$

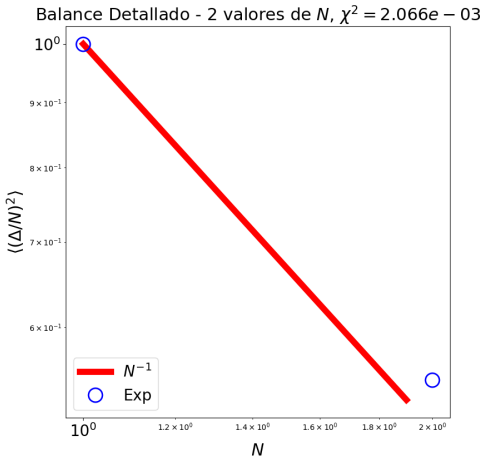


(b) $\left\langle \left(\Delta^{(i,j)}/N^{(i,j)}\right)^2 \right\rangle$ en sistema CG-CML de 4×4 sitios con una evolución temporal de 4×10^5 iteraciones y semilla $s = 37596202183728265 29$

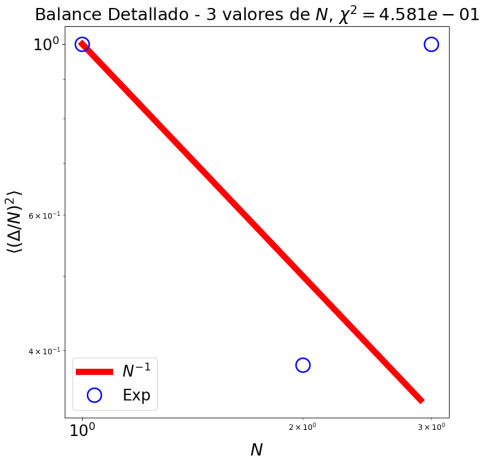
Figura 4.16: Dependencia de $\left\langle \left(\Delta^{(i,j)}/N^{(i,j)}\right)^2 \right\rangle$ con el total de transiciones entre pares de estados (i, j) disponibles a sistemas CG-CML de 4×4 sitios, elaborados a partir de rejillas de mapeo acoplado de longitud $L = 32$, tamaño de grano $G = 8$ y acoplamiento $g = 0.204$. Los círculos azules denotan los promedios $\left\langle \left(\Delta^{(i,j)}/N^{(i,j)}\right)^2 \right\rangle$ para las diferentes mediciones de $\Delta^{(i,j)}$ para valores similares del total de transiciones $N^{(i,j)}$ y la línea roja marca la función N^{-1}. Se consideró $\Delta T = 100$ para el cálculo de los promedios temporales. La tasa de crecimiento de la función exponencial usada para definir los *bins* que agrupan mediciones de $\Delta^{(i,j)}$ fue igual a 1.

Adicionalmente, con miras a profundizar en la naturaleza de los resultados discutidos, aplicamos la simulación a rejillas "a grano grueso" de 4×4 sitios derivadas de sistemas CML con un acoplamiento nulo ($g = 0$). La Figura 4.17 muestra las gráficas resultantes. Aquí es necesario reiterar que χ^2 no determina unívocamente la presencia de balance detallado. La Gráfica 4.17a tiene asociado el valor mínimo de dicho parámetro entre los resultados obtenidos para sistemas CG-CML elaborados a partir de rejillas de mapeo acoplado con $L = 32$ y $G = 8$. Empero, el número de valores diferentes para N correspondiente a tal gráfica también es el menor de todos; el total de transiciones N entre los pares de estados que visita el sistema CG-CML durante su evolución es igual a 1 o 2, es decir, no hay un par de estados que haya experimentado más de 2 transiciones entre sí dado un tiempo de evolución de 2×10^5 iteraciones. De hecho, el total de diferentes valores para N sólo asciende a 3 al duplicar la duración de la evolción temporal de sistema, como lo muestra la Gráfica 4.17b[12]. Por lo tanto, concluimos que un acoplamiento nulo entre sitios de una rejilla de mapeo acoplado no conduce a balance detallado en su respectivo sistema CG-CML.

[12]El *binning* exponencial hecho por el código en el Apéndice I pone de relieve las discrepancias entre los resultados empíricos y las predicciones teóricas (Figura 4.18), pero la falta de una justificación detrás de tal operación ofusca la importancia de ello.

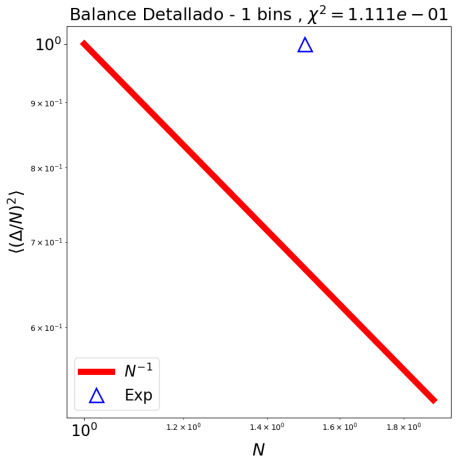


(a) $\left\langle \left(\Delta^{(i,j)}/N^{(i,j)}\right)^2 \right\rangle$ en sistema CG-CML de 4×4 sitios con una evolución temporal de 2×10^5 iteraciones y semilla $s = 6197905548778816112$

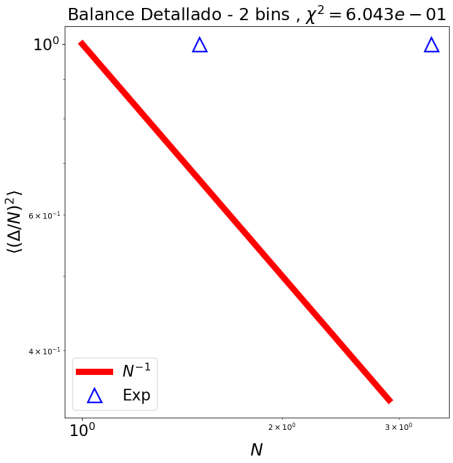


(b) $\left\langle \left(\Delta^{(i,j)}/N^{(i,j)}\right)^2 \right\rangle$ en sistema CG-CML de 4×4 sitios con una evolución temporal de 4×10^5 iteraciones y semilla $s = 4523530975986176005$

Figura 4.17: Comportamiento de $\left\langle \left(\Delta^{(i,j)}/N^{(i,j)}\right)^2 \right\rangle$ con respecto al total de transiciones entre pares de estados (i, j) disponibles a sistemas CG-CML de 4×4 sitios, elaborados a partir de rejillas de mapeo acoplado de longitud $L = 32$, tamaño de grano $G = 8$ y acoplamiento $g = 0$. Los círculos azules marcan los valores del promedio $\left\langle \left(\Delta^{(i,j)}/N^{(i,j)}\right)^2 \right\rangle$ para las diferentes mediciones de $\Delta^{(i,j)}$ dado un total de transiciones $N^{(i,j)} = N$, mientras que la línea roja representa N^{-1}, el comportamiento esperado de un sistema con balance detallado. Se consideró $\Delta T = 100$ para el cálculo de los promedios temporales.



(a) $\left\langle \left(\Delta^{(i,j)}/N^{(i,j)}\right)^2 \right\rangle$ sobre *bins* en sistema CG-CML de 4×4 sitios con una evolución temporal de 2×10^5 iteraciones y semilla $s = 6197905548778816112$



(b) $\left\langle \left(\Delta^{(i,j)}/N^{(i,j)}\right)^2 \right\rangle$ en sistema CG-CML de 4×4 sitios con una evolución temporal de 4×10^5 iteraciones y semilla $s = 45235309759861760 05$

Figura 4.18: Dependencia de $\left\langle \left(\Delta^{(i,j)}/N^{(i,j)}\right)^2 \right\rangle$ con el total de transiciones entre pares de estados (i, j) disponibles a sistemas CG-CML de 4×4 sitios, elaborados a partir de rejillas de mapeo acoplado de longitud $L = 32$, tamaño de grano $G = 8$ y acoplamiento $g = 0.000$. Los círculos azules denotan los promedios $\left\langle \left(\Delta^{(i,j)}/N^{(i,j)}\right)^2 \right\rangle$ para las diferentes mediciones de $\Delta^{(i,j)}$ para valores similares del total de transiciones $N^{(i,j)}$ y la línea roja marca la función N^{-1}. Se consideró $\Delta T = 100$ para el cálculo de los promedios temporales. La tasa de crecimiento de la función exponencial usada para definir los *bins* que agrupan mediciones de $\Delta^{(i,j)}$ fue igual a 1.

4.5. Ensamble de Gibbs

El programa que aparece en el Apéndice J obtiene los valores óptimos de los coeficientes $\alpha\,(d)$ que marcan el acoplamiento entre sitios separados una distancia d en una rejilla a grano grueso.

4.5.1. Detalles Técnicos Computacionales sobre la prueba de Ensamble de Gibbs

Antes de efectuar cualquier cálculo, el código pide al usuario los valores de los parámetros contemplados en la simulación de balance detallado, junto con la semilla del generador de números aleatorios usada en dicho procedimiento, un número natural m y una constante real c (165-177). Tras escoger un entero positivo al azar como semilla del generador de número aleatorios idénticamente a los programas descritos en las secciones previas (178-179), los datos recopilados son usados en primera instancia para identificar el archivo con la evolución temporal de la rejilla "a grano grueso" y guardar tal información en un arreglo de Python (180-184). Luego, el programa genera y almacena un histograma con dicho arreglo (186-193). Al ajustar el número de *bins* al total de estados disponibles a la rejilla, limitar el rango al de los estados

y asignar un valor `True` a la opción `density`, el histograma constituye una aproximación a la distribución de probabilidad de los estados del sistema.

El código utiliza después los valores asignados a las propiedades de la rejilla "a grano grueso" para construir una gráfica del sistema dispuesto en el estado 0 ($\tilde{u}^T_{\vec{x}} = -1\ \forall \vec{x}$) e inicializa un diccionario para almacenar las sumas de los productos $\tilde{u}^T_{\vec{x}}\tilde{u}^T_{\vec{y}}$ sobre todos los pares de sitios $(\vec{x}, \vec{y})$ con una separación d entre ellos, considerando las diferentes distancias posibles entre dos sitios cualesquiera en la rejilla (195-199) (Figura 4.19). Denotando tales sumas como sPE(d), el Hamiltoniano $\mathcal{H}$ podría reformularse como

$$\begin{aligned}\mathcal{H} = \sum_{\vec{x},\vec{y}} \alpha\left(d\right) \tilde{u}^T_{\vec{x}}\tilde{u}^T_{\vec{y}} &= \sum_{d} \alpha(d) \cdot \left(\sum_{\vec{x},\vec{y}:|\vec{x}-\vec{y}|=d} \tilde{u}^T_{\vec{x}}\tilde{u}^T_{\vec{y}} \right) \\ &= \sum_{d} \alpha(d) \cdot \mathrm{sPE}(d) \end{aligned} \tag{4.3}$$

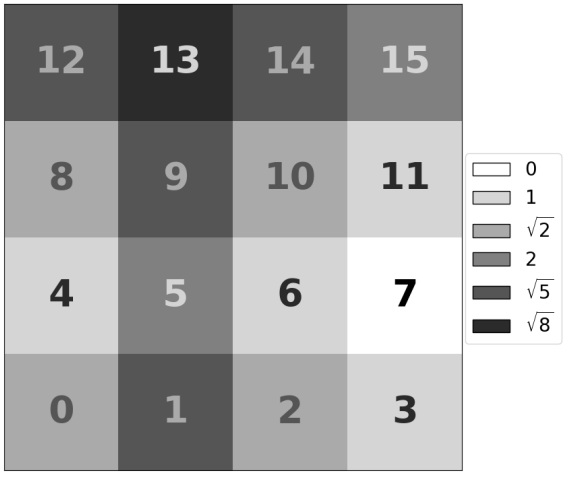


Figura 4.19: Ilustración de las distancias entre el sitio $n(\vec{x}_0) = 7$ y los demás elementos de una rejilla "a grano grueso" conformada por 4×4 sitios, bajo condiciones de frontera periódicas. Los colores de los números $n(\vec{x}_0)$ no tienen significado alguno y sólo fueron seleccionados para generar contraste y facilitar su lectura. Para este sistema CG-CML, la energía $\mathcal{H}$ (Ecuación 4.3) contemplaría cinco coeficientes $\alpha(d)$ distintos.

El ciclo `for` que inicia en la línea 201 calcula las sumas sPE(d) para cada estado disponible al sistema. Una vez configurada la gráfica en un estado específico con la función `def_edo` (definida en 52-66), el programa estima todos los productos $\tilde{u}^T_{\vec{x}}\tilde{u}^T_{\vec{y}}$, los agrupa en listas de acuerdo a la distancia entre sus respectivos sitios y guarda cada lista como una entrada en un diccionario adicional (205-217). Después, obtiene la suma de los elementos de cada lista (los factores sPE(d)) y agrega el diccionario con dichas sumas como un elemento del diccionario definido antes del *loop*, utilizando la iteración del ciclo (equivalente al número de estado de la rejilla)

como clave (218-224). Tras llevar la gráfica al estado 0 (225), el proceso anterior se repite para el siguiente estado. El diccionario que contiene los conjuntos de sumas sPE(d) (que denominamos proto-energías) para todos los estados accesibles a la rejilla es posteriormente almacenado en un archivo de texto (226-230). El código procede entonces a obtener una lista de las proto-energías disponibles al sistema sin repeticiones y conservar tal versión reducida en una lista y otro archivo .txt (232-244).

Una vez concluido el procedimiento previo, el código junta las proto-energías del sistema con sus correspondientes probabilidades: en un barrido del conjunto con las diferentes proto-energías (246-262), el programa recopila los números de estados asociados a una proto-energía particular examinando el diccionario que contiene todas las proto-energías disponibles (250-256), identifica la cantidad de estados ligados a la proto-energía seleccionada (257), calcula su probabilidad como la suma de las probabilidades de sus respectivos estados (258-261) y guarda en una lista (inicializada en 248) la proto-energía, su probabilidad y el total de estados con tal proto-energía (262). Al terminar la aplicación de este algoritmo sobre todas las proto-energías, la lista es almacenada en un archivo de texto (263-267), tras lo cual sus elementos son reordenados en un orden decreciente de probabilidad y guardados en un archivo distinto con el mismo formato (269-292). Tal acomodo para confirmar su funcionamiento fue incluido durante el desarrollo del código y optamos por preservarlo para facilitar la revisión de los resultados de otros procesos descritos más adelante.

A continuación, el código produce una lista ordenada de las distancias d existentes para cualquier par de sitios en la rejilla "a grano grueso" (295-296) mediante la función `list_dists` (68-77), construye una lista de claves con dichas distancias codificadas (297-300) y define una restricción sobre las energías aceptables para el sistema (302-319).

El segmento del programa comprendido entre las líneas 321-709 implementa las cantidades previamente obtenidas para hallar las magnitudes de los coeficientes $\alpha(d)$ que producen una distribución de energías (dadas por el Hamiltoniano $\mathcal{H}$ en la Ecuación 4.3) equivalente (o al menos similar) a una distribución de Boltzmann. Para cumplir tal meta, el código intenta minimizar la suma (sobre las diferentes energías disponibles al sistema) de los cuadrados de las diferencias entre dos probabilidades:

- La probabilidad "empírica" de cada energía, calculada como la suma de las probabilidades de los estados correspondientes a su respectiva proto-energía. Cabe resaltar que la probabilidad de un estado proviene de la distribución probabilística (aproximada) de estados elaborada a partir de la evolución temporal de la rejilla "a grano grueso".
- La probabilidad "teórica" de cada energía conforme a una distribución de Boltzmann. Las energías son estimadas de acuerdo al Hamiltoniano $\mathcal{H}$ asumiendo un conjunto de valores específicos para los coeficientes $\alpha(d)$.

Dado un conjunto de valores iniciales para los coeficientes $\alpha(d)$ (escogidos con aleatoriedad entre $[-1, 1]$), el programa calcula las energías disponibles al sistema y sus correspondientes probabilidades "teóricas" con la función `prob_boltzmann` (definida en 79-103), estima la suma χ^2 de los cuadrados de las diferencias entre los dos tipos de probabilidades destacados antes mediante la función `min_cuadrados` (105-111), modifica los valores $\alpha(d)$ y repite el procedimiento anterior hasta conseguir un mínimo para χ^2[13]. Una vez concluida, la optimización anterior es

[13]Las últimas dos partes de este algoritmo son realizadas por la función `minimize` del paquete

repetida $m-1$ veces empleando *ansatzes* distintos para los valores iniciales de $\alpha(d)$, donde m es el entero positivo proporcionado por el usuario al inicio de la simulación.

Una vez completado el proceso detallado en el párrafo precedente, el código almacena los diferentes *ansatzes* y sus correspondientes valores para los coeficientes $\alpha(d)$. Luego, identifica el conjunto "óptimo" de coeficientes $\alpha(d)$[14] y obtiene la distribución de probabilidad para las energías y función de partición Z asociadas a él. Posteriormente, el programa calcula las energías disponibles al sistema con los coeficientes $\alpha(d)$ "óptimos" y produce las energías y probabilidades de Boltzmann para cada estado del sistema CG-CML. Finalmente, las entradas de los arreglos con las energías y las distribuciones probabilísticas teórica y empírica (tanto para las energías como los estados) son reordenadas con la función `ordenador` (143-160) para facilitar la realización de su gráfica.

El programa contempla cuatro esquemas de minimización que difieren en dos aspectos:

- **El método suministrado a `minimize` para determinar el mínimo de la suma de cuadrados χ^2**
  Es posible configurar el método numérico que implementa la función `minimize` para hallar puntos mínimos. En miras de reducir el espacio de los coeficientes $\alpha(d)$, optamos por usar el método `trust-constr` para limitar las energías del sistema a valores no negativos (conforme a la Figura 3.4). Adicionalmente, usamos la función `minimize` en otros casos sin especificar un método numérico. De acuerdo a la documentación de `scipy`, tal decisión implica que la función elige entre los métodos `BFGS`, `L-BFGS-B` y `SLSQP` según la naturaleza de las fronteras de los dominios de las variables y demás restricciones (lineales o no lineales).

- **La simultaneidad del ajuste de los coeficientes $\alpha(d)$**
  La función `minize` puede trabajar sobre múltiples variables a la vez. Si bien usamos la función de este modo para algunos casos, también consideramos determinar los coeficientes $\alpha(d)$ uno a la vez, en orden creciente de la distancia d. El programa completa esta tarea con un ciclo `for` que aplica la función `minimze` sobre un coeficiente específico en una iteración con un dominio sin fronteras y fija tal valor para las subsecuentes iteraciones. Por ejemplo, en la primera iteración de dicho ciclo, la función `minimize` puede asignar valores en el intervalo $(-\infty, \infty)$ al coeficiente $\alpha(1)$ pero considera que los otros coeficientes únicamente pueden ser equivalentes a su respectivo valor inicial. En la segunda iteración, el dominio del coeficiente $\alpha(1)$ queda limitado al valor que la función `minimize` determinó en la iteración previa, el coeficiente $\alpha(\sqrt{2})$ queda libre de cualquier restricción y los demás coeficientes permanecen fijos durante la operación de la función `minimize`.

Los cuatros esquemas de minimización implementados en el programa son los siguientes:

- **Esquema local 0** (325-414): La función `minimize` es usada sin especificar método y los coeficientes $\alpha(d)$ son determinados simultáneamente. La ausencia de cualquier restricción implica que `minimize` selecciona el método `BFGS`.

- **Esquema local 1** (417-503): La función `minimize` utiliza el método `trust-constr` y los coeficientes $\alpha(d)$ son examinados al mismo tiempo.

`scipy.optimize`.

[14] En este contexto, "óptimo" describe al conjunto de valores $\alpha(d)$ que minimizan χ^2.

- **Esquema gradual 0** (506-606): La función `minimize` es implementada sin definir explícitamente un método para minimizar la suma de cuadrados y los coeficientes $\alpha(d)$ son determinados de uno en uno. Como el ajuste gradual de los coeficientes impone límites en el dominio de tales variables, la función escoge el método `L-BFGS-B`.

- **Esquema gradual 1** (609-709): La función `minimize` opera bajo el método `trust-constr` en el ajuste gradual de los coeficientes $\alpha(d)$.

4.5.2. Resultados de la prueba de Ensamble de Gibbs

Al igual que en el caso de balance detallado, optamos inicialmente por determinar el correcto funcionamiento del programa tratando de recuperar una distribución de Boltzmann a partir de la evolución temporal de las rejillas de Ising mencionadas en el apartado previo. Implementamos para ello el código en las líneas 1-240 y 370-682 en el Apéndice K, que corresponde a una versión del programa recién descrito adaptada a las características propias del modelo de Ising. La Figura 4.20 exhibe los resultados derivados de las rejillas de Ising de 4×4 sitios examinadas en la Sección 4.4 al usar la función `minimize`[15] y muestra que una evolución temporal más larga genera una distribución de probabilidad más parecida a la distribución de Gibbs teórica, aunque los acoplamientos ajustados son sorprendentemente acertados para los diferentes tiempos de evolución (Cuadro 4.1). Volvimos a realizar las simulaciones con otros acoplamientos J, consiguiendo al final valores J_{fit} equivalentes (ver Figura 4.21 y Cuadro 4.2). Así, el ejercicio anterior garantiza que el código opera en la forma esperada.

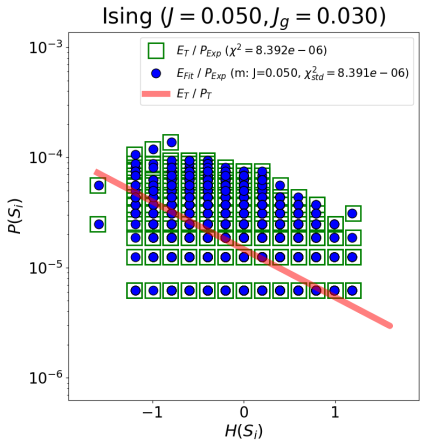
Número (promedio) de rondas por sitio	J_{fit}
10^4	4.951011763904211022E-02
10^5	4.881398722865511186E-02
10^6	5.003199153920163261E-02

Cuadro 4.1: Acoplamientos J_{fit} ajustados a distribuciones de probabilidad asociadas a rejillas de Ising de 4×4 sitios con diferentes tiempos de evolución, usando la función `minimize`

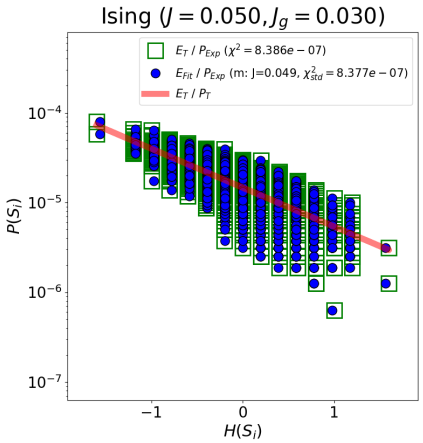
J	J_g	J_{fit}	e_{rel} (%)
0.01	0.016	9.799720299536849796E-03	-2.043728742672650
0.02	0.040	1.994719143444910817E-02	-0.264741859646958
0.03	0.025	3.004100494134295113E-02	0.136496570014769
0.04	0.100	3.999099802265290021E-02	-0.022510009232581
0.05	0.030	5.003199153920163261E-02	0.063942166236834

Cuadro 4.2: Acoplamientos J_{fit} ajustados a distribuciones de probabilidades obtenidas a partir de la evolución temporal de rejillas de Ising de 4×4 sitios con diferentes acoplamientos J, considerando distintos valores iniciales J_g al usar `minimize` y una evolución temporal de 1.6×10^7 iteraciones. junto con errores relativos $e_{rel} = (J - J_{fit})/J_{fit}$

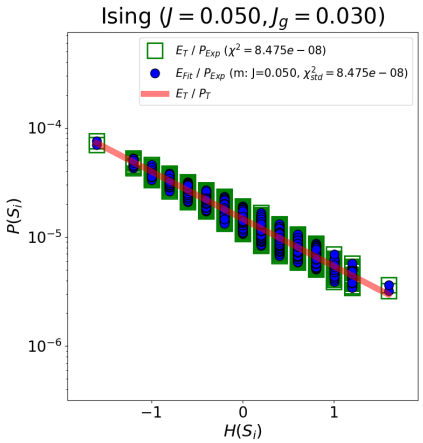
[15] El programa en el Apéndice K también usa `curve_fit` y `minimize_scalar` para hallar el acoplamiento J que ofrece el mejor ajuste a los datos experimentales. Los valores determinados con tales métodos son sumamente parecidos al acoplamiento obtenido con `minimize` y únicamente difieren de él a partir de la cuarta cifra significativa.



(a) Distribuciones de probabilidad (teórica y experimental) de una rejilla de Ising de 4×4 sitios, considerando una evolución temporal de 1.6×10^5 iteraciones y semilla $s = 6229836497689036069$



(b) Distribuciones de probabilidad (teórica y experimental) de una rejilla de Ising de 4×4 sitios, considerando una evolución temporal de 1.6×10^6 iteraciones y semilla $s = 1016131477794588868$



(c) Distribuciones de probabilidad (teórica y experimental) de una rejilla de Ising de 4×4 sitios, considerando una evolución temporal de 1.6×10^7 iteraciones y semilla $s = 7450712230650374706$

Figura 4.20: Distribución de probabilidades $\mathcal{P}(S_i)$ en términos de las energías $\mathcal{H}(S_i)$ en rejillas de Ising de 4×4 sitios con acoplamiento $J = 0.05$ entre primeros vecinos. La distribución de Boltzmann asociada al sistema aparece como una recta roja, los cuadradados de bordes verdes representan las energías teóricas con sus respectivas probabilidades experimentales y las coordenadas de los círculos azules corresponden a las energías y las probabilidades ajustadas. El *ansatz* para el acoplamiento proporcionado a la simulación es $J_g = 0.03$ y la función implementada para determinar el valor óptimo de J fue `minimize`.

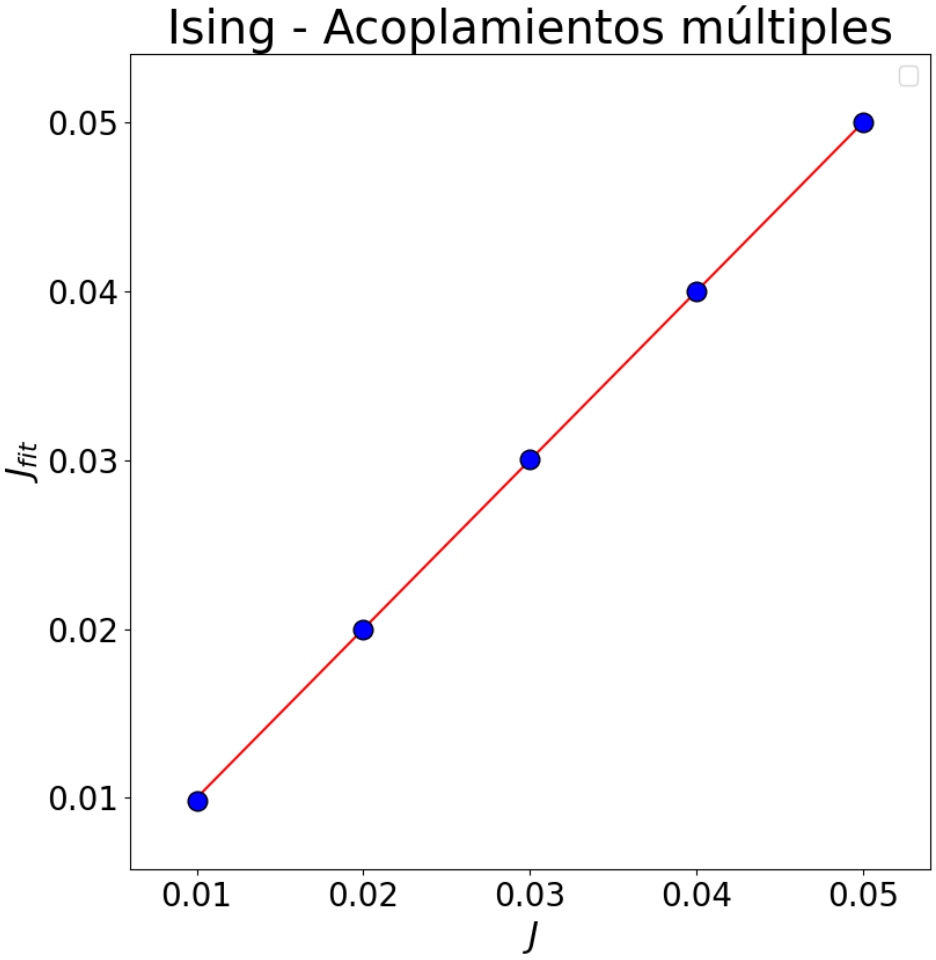


Figura 4.21: Relación entre acoplamientos J y los valores ajustados J_{fit} a rejillas de Ising de 4×4 sitios, aplicando el algoritmo de Metropolis 10^6 veces por sitio.

Posteriormente, hicimos simulaciones sobre las cuatro rejillas "a grano grueso" de 4×4 sitios con acoplamiento $g = 0.204$ analizadas en la Sección 4.4 (construidas a partir de sistemas CML de longitudes y tamaños de grano $\{L = 8, G = 2\}$ y $\{L = 32, G = 8\}$). Los resultados obtenidos con los cuatro esquemas de minimización en las primeras dos secciones del Apéndice L.3; en particular, los productos de las simulaciones bajo el esquema de minimización "Local 0" para rejillas con una evolución temporal de 4×10^5 iteraciones quedan ilustrados en las Figuras 4.22 y 4.23, junto a sus respectivos valores para los coeficientes $\alpha(d)$ y la dispersión χ^2 (Cuadros 4.3 y 4.4)[16].

Las Figuras L.2, L.4, L.5 y L.7 revelan inmediatamente que la restricción de las energías $\mathcal{H}(S_i)$ a valores estrictamente positivos es inadecuada; las distribuciones asociadas a los coeficientes $\alpha(d)$ que la función `minimize` califica como óptimos bajo el método `trust-constr` cuentan con valores para la dispersión χ^2 al menos tres órdenes de magnitud superiores a los

[16]Los resultados de estas mismas simulaciones para sistemas con tiempos de evolución de 2×10^5 iteraciones aparecen en las Secciones L.1 y L.2

de las distribuciones obtenidas al implementar el método `BFGS`. Las mismas gráficas también prueban que las diferencias entre los esquemas de minimización de tipo "Local" (que ajustan los coeficientes simultáneamente) y los de tipo "Gradual" (que ajustan los coeficientes de uno en uno) que usan el mismo método de optimización son mínimas y llevan a resultados similares.

Asimismo, las simulaciones que usan `minimize` libre de restricciones sobre las energías muestran que ambos sistemas CG-CML, sin importar el tiempo de evolución o las dimensiones de la rejilla "fina" subyacente, cuentan con una distribución de Gibbs. Sorprendentemente, las distribuciones de probabilidades de la rejilla "a grano grueso" correspondientes a sistemas CML de 8×8 sitios (los cuales no exhiben balance detallado) son mejores ajustes que las distribuciones de sistemas desarrollados a partir de rejillas de mapeo acoplado de longitud $L = 32$. Ninguno de los conjuntos de coeficientes $\alpha(d)$ decrece de forma monótona con la distancia d entre sitios como los valores reportados en [5] (Cuadro 3.1), aunque la relación entre los signos de los valores asociados a los sistemas CML de 32×32 sitios es la misma (Cuadros L.2 y 4.4) y el primer coeficiente también es el más grande de todos para las cuatro rejillas "a grano grueso" aquí examinadas.

El origen más probable de las discrepancias entre los puntos empíricos y las distribuciones de Gibbs obtenidas bajo los esquemas "Local 0" y "Gradual 0" es un tiempo de evolución muy corto, lo cual se refleja principalmente en la línea horizontal de círculos azules en sus respectivas gráficas. Tales puntos representan estados con las energías más grandes y consecuentemente las menores probabilidades, las cuales posiblemente fueron sobreestimadas como resultado de una evolución temporal breve. Incrementar el número de iteraciones debería producir entonces mejores ajustes. Sin embargo, los resultados aquí presentados sugieren que dicho aumento debería ser de órdenes de magnitud, ya que duplicar el tiempo de evolución fue insuficiente para disminuir la dispersión.

Realizamos también simulaciones sobre las rejillas "a grano grueso" definidas a partir de sistemas CML de longitud $L = 32$, tamaño de grano $G = 8$ y acoplamiento nulo ($g = 0$). Los resultados correspondientes al conjunto de coeficientes $\alpha(d)$ determinados bajo el esquema de minimización "Local 0" (así como los coeficientes en sí) dada una evolución temporal de 4×10^5 iteraciones aparecen en la Figura 4.24 y el Cuadro 4.5. Las gráficas y valores para los coeficientes $\alpha(d)$ asociados a otros esquemas de minimización y tiempos de evolución se encuentran en la tercera sección del Apéndice L.3.

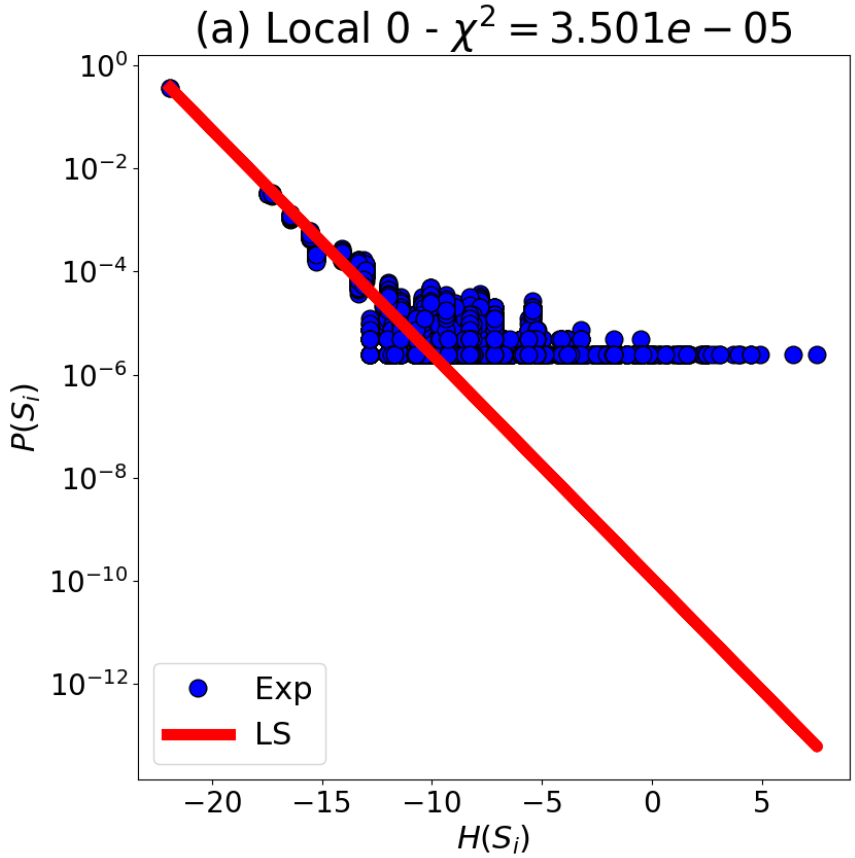


Figura 4.22: Distribución de probabilidades $\mathcal{P}(\mathcal{S}_i)$ de los estados S_i en términos de las energías $\mathcal{H}(S_i)$ disponibles a un sistema CG-CML de 4×4 sitios con una evolución temporal de 4×10^5 iteraciones y semilla $s = 5617058624460882590$, usando el esquema de minimización local 0. Tal sistema fue generado a partir de una rejilla de mapeo acoplado con longitud $L = 8$, grano $G = 2$ y acoplamiento $g = 0.204$. Los puntos azules corresponden a las mediciones de las probabilidades de cada energía tras el ajuste y la recta roja marca la distribución de Boltzmann asociada a los coeficientes $\alpha(d)$ ajustados.

Método	$\alpha(1)$	$\alpha(\sqrt{2})$	$\alpha(2)$	$\alpha(\sqrt{5})$	$\alpha(\sqrt{8})$	χ^2
Local 0	-1.1511E+00	3.9722E-01	1.4520E-01	5.4272E-02	-2.2865E-01	3.5009E-05
Gradual 0	-1.1511E+00	3.9722E-01	1.4521E-01	5.4275E-02	-2.2866E-01	3.5009E-05
Local 1	1.2850E-11	2.4701E-11	4.5447E-10	4.6570E-11	2.0879E-10	2.6628E-01
Gradual 1	5.1955E-11	9.7101E-11	3.6635E-10	-7.5789E-11	-2.8271E-11	2.6628E-01

Cuadro 4.3: Acoplamientos $\alpha(d)$ ajustados a distribución experimental de probabilidades $\mathcal{P}(S_i)$ de sistema CG-CML de 4×4 sitios con una evolución temporal de 4×10^5 iteraciones, elaborado a partir de una rejilla de mapeo acoplado de longitud $L = 8$, tamaño de grano $G = 2$ y acoplamiento $g = 0.204$

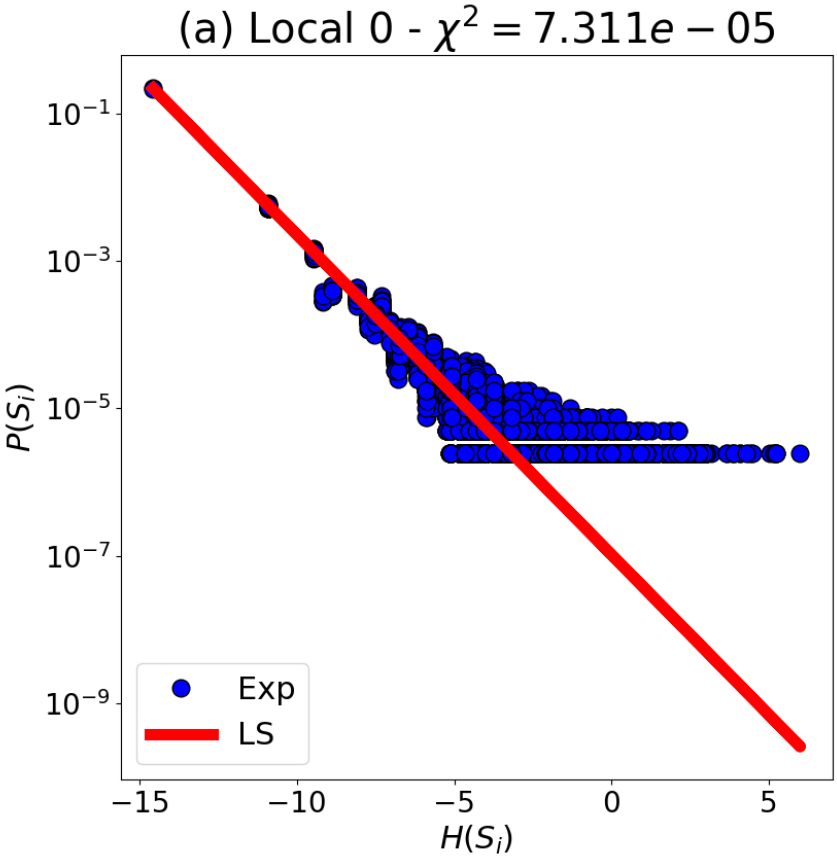


Figura 4.23: Distribución de probabilidades $\mathcal{P}(\mathcal{S}_i)$ de los estados S_i en términos de las energías $\mathcal{H}(S_i)$ disponibles a un sistema CG-CML de 4×4 sitios con una evolución temporal de 4×10^5 iteraciones y semilla $s = 3759620218372826529$, usando el esquema de minimización local 0. Tal sistema fue generado a partir de una rejilla de mapeo acoplado con longitud $L = 32$, grano $G = 8$ y acoplamiento $g = 0.204$. Los puntos azules corresponden a las mediciones de las probabilidades de cada energía tras el ajuste y la recta roja marca la distribución de Boltzmann asociada a los coeficientes $\alpha(d)$ ajustados.

Método	$\alpha(1)$	$\alpha(\sqrt{2})$	$\alpha(2)$	$\alpha(\sqrt{5})$	$\alpha(\sqrt{8})$	χ^2
Local 0	-5.4993E-01	-1.0756E-02	1.3295E-01	5.5446E-02	-6.5080E-02	7.3110E-05
Gradual 0	-5.4992E-01	-1.0763E-02	1.3293E-01	5.5461E-02	-6.5098E-02	7.3111E-05
Local 1	-1.3295E-11	-2.8606E-11	5.3438E-10	-1.6795E-11	4.9536E-11	9.5926E-02
Gradual 1	-2.5543E-11	5.9742E-11	4.1907E-10	-1.8729E-11	1.2069E-10	9.5926E-02

Cuadro 4.4: Acoplamientos $\alpha(d)$ ajustados a distribución experimental de probabilidades $\mathcal{P}(S_i)$ de sistema CG-CML de 4×4 sitios con una evolución temporal de 4×10^5 iteraciones, elaborado a partir de una rejilla de mapeo acoplado de longitud $L = 32$, tamaño de grano $G = 8$ y acoplamiento $g = 0.204$

Observamos que un acoplamiento nulo entre sitios de una rejilla "fina" no otorga una distribución de Gibbs a los estados disponibles al sistema CG-CML asociado. La dispersión χ^2 es

significativamente menor que los valores de tal parámetro en las simulaciones sobre rejillas "a grano grueso" elaboradas a partir de sistemas CML con un acoplamiento distinto de cero, pero esto es producto del limitado rango de las probabilidades experimentales; las probabilidades de cada estado disponible únicamente abarcan valores desde 2×10^{-6} hasta 2×10^{-5}, en contraste con los seis órdenes de magnitud que cubren las probabilidades empíricas en las otras simulaciones. La esporádica ocurrencia de cada estado en la rejilla "a grano grueso" asociada a $g = 0$ resulta entonces en valores ínfimos para todos los coeficientes $\alpha(d)$, siendo la magnitud más grande de cualquiera de dichos coeficientes del orden de 10^{-3} (ver Cuadro L.3). Así, la función exponencial ajustada arroja probabilidades suficientemente pequeñas para alcanzar una dispersión χ^2 mínima, incluso si esto implica un crecimiento de la probabilidad con respecto a la energía (ver Gráficas (b) en las Figuras L.8 y L.11). Al final, las energías de los estados disponibles al sistema CG-CML obtenido al tomar un acoplamiento nulo no tienen el efecto sobre sus correspondientes probabilidaes que es de esperarse en un ensamble canónico.

En aras de esclarecer por qué Egolf únicamente contempla los primeros cuatro coeficientes $\alpha(d)$ al calcular la energía $\mathcal{H}(S_i)$ cuando existen sitios en una rejilla "a grano grueso" de 4×4 sitios con una distancia $d = \sqrt{8}$ (ver Figura 4.19), repetimos las simulaciones descritas anteriormente considerando diferentes números de coeficientes. Usamos para tal tarea el programa en el Apéndice M, que consiste esencialmente en un *loop* con el código descrito previamente para múltiples cantidades de coeficientes $\alpha(d)$. El resultado de esta clase de simulación en el sistema CG-CML derivado de una rejilla de mapeo acoplado de longitud $L = 32$, tamaño de grano $G = 8$, acoplamiento $g = 0.204$ y evolución temporal de 4×10^5 iteraciones está ilustrado en la Figura 4.25[16]. Los cambios en la dispersión χ^2 ocasionados por la inclusión de más coeficientes $\alpha(d)$ en la energía de dicho sistema aparecen en la Figura 4.26[16]. Las gráficas en ambas Figuras revelan que las discrepancias entre las probabilidades empíricas de los estados y la distribución de Gibbs ajustada a ellas son mínimas al tomar en cuenta tres coeficientes $\alpha(d)$, aunque los valores de χ^2 en las simulaciones que consideran cuatro o cinco coeficientes son del mismo orden y las diferencias en la magnitud de la dispersión son muy pequeñas (en relación a los valores obtenidos al usar únicamente uno o dos coeficientes). La naturaleza de estos resultados impide determinar si el uso de cuatro coeficientes $\alpha(d)$ es preferible al de cinco coeficientes; la dispersión χ^2 en las distribuciones de Gibbs correspondientes adquiere valores similares y ambos ajustes son menos acertados que la distribución de probabilidad obtenida con tres coeficientes.

[16]Los resultados para el sistema CG-CML asociado a una rejilla "fina" de longitud $L = 32$, tamaño de grano $G = 8$, acoplamiento $g = 0.204$ y evolución temporal de 2×10^5 iteraciones fueron anexados en el Apéndice .

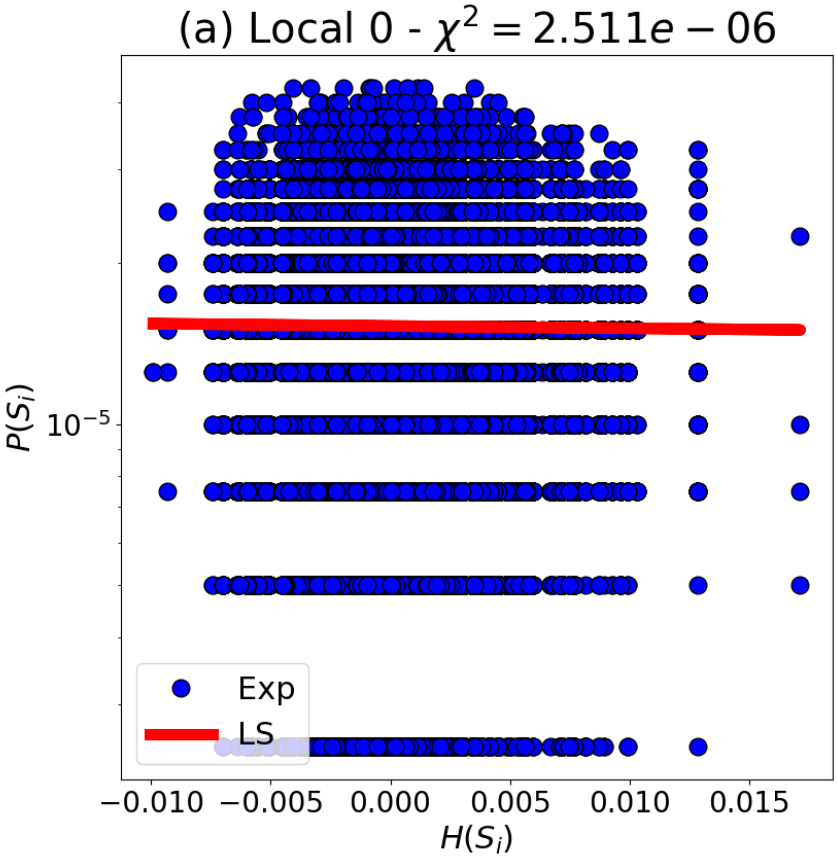
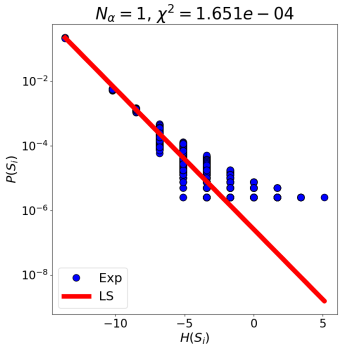


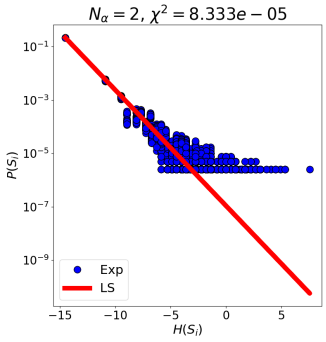
Figura 4.24: Distribución de probabilidades $\mathcal{P}(\mathcal{S}_i)$ de los estados S_i en términos de las energías $\mathcal{H}(S_i)$ disponibles a un sistema CG-CML de 4×4 sitios con una evolución temporal de 4×10^5 iteraciones y semilla $s = 4523530975986176005$, usando el esquema de minimización local 0. Tal sistema fue generado a partir de una rejilla de mapeo acoplado con longitud $L = 32$, grano $G = 8$ y acoplamiento $g = 0$. Los puntos azules corresponden a las mediciones de las probabilidades de cada energía tras el ajuste y la recta roja marca la distribución de Boltzmann asociada a los coeficientes $\alpha(d)$ ajustados.

Método	$\alpha(1)$	$\alpha(\sqrt{2})$	$\alpha(2)$	$\alpha(\sqrt{5})$	$\alpha(\sqrt{8})$	χ^2
Local 0	9.7275E-05	-3.3654E-04	1.7901E-04	-2.6840E-04	4.3505E-04	2.5114E-06
Gradual 0	9.7805E-05	-3.3264E-04	1.8018E-04	-2.6645E-04	4.3180E-04	2.5114E-06
Local 1	1.9775E-11	1.6929E-10	8.9495E-10	8.8026E-12	8.6778E-10	2.5113E-06
Gradual 1	-1.5430E-11	7.6272E-11	8.8104E-10	-4.2240E-11	8.0930E-10	2.5113E-06

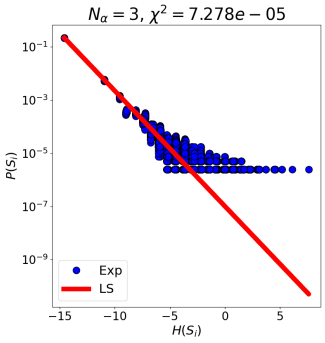
Cuadro 4.5: Acoplamientos $\alpha(d)$ ajustados a distribución experimental de probabilidades $\mathcal{P}(S_i)$ de sistema CG-CML de 4×4 sitios con una evolución temporal de 4×10^5 iteraciones, elaborado a partir de una rejilla de mapeo acoplado de longitud $L = 32$, tamaño de grano $G = 8$ y acoplamiento $g = 0$



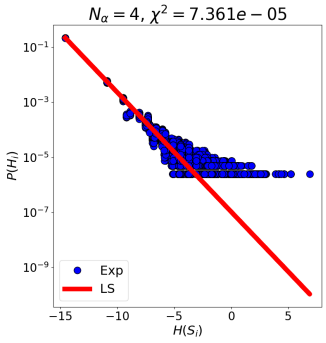
(a) Distribuciones de probabilidad de un sistema CG-CML de 4×4 sitios, considerando $N_\alpha = 1$ coeficiente para la energía $\mathcal{H}(S_i)$



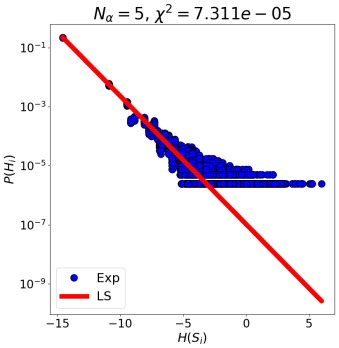
(b) Distribuciones de probabilidad de un sistema CG-CML de 4×4 sitios, considerando $N_\alpha = 2$ coeficientes para la energía $\mathcal{H}(S_i)$



(c) Distribuciones de probabilidad de un sistema CG-CML de 4×4 sitios, considerando $N_\alpha = 3$ coeficientes para la energía $\mathcal{H}(S_i)$



(d) Distribuciones de probabilidad de un sistema CG-CML de 4×4 sitios, considerando $N_\alpha = 4$ coeficientes para la energía $\mathcal{H}(S_i)$



(e) Distribuciones de probabilidad de un sistema CG-CML de 4×4 sitios, considerando $N_\alpha = 5$ coeficientes para la energía $\mathcal{H}(S_i)$

Figura 4.25: Distribuciones de probabilidad $\mathcal{P}(S_i)$ obtenidas al considerar diferentes números N_α de coeficientes $\alpha(d)$ en el cálculo de la energía de un sistema CG-CML de 4×4 sitios con una evolución temporal de 4×10^5 iteraciones y semilla $s = 37596202183728265 29$, derivado de una rejilla de mapeo acoplado de longitud $L = 32$, $G = 8$ y $g = 0.204$

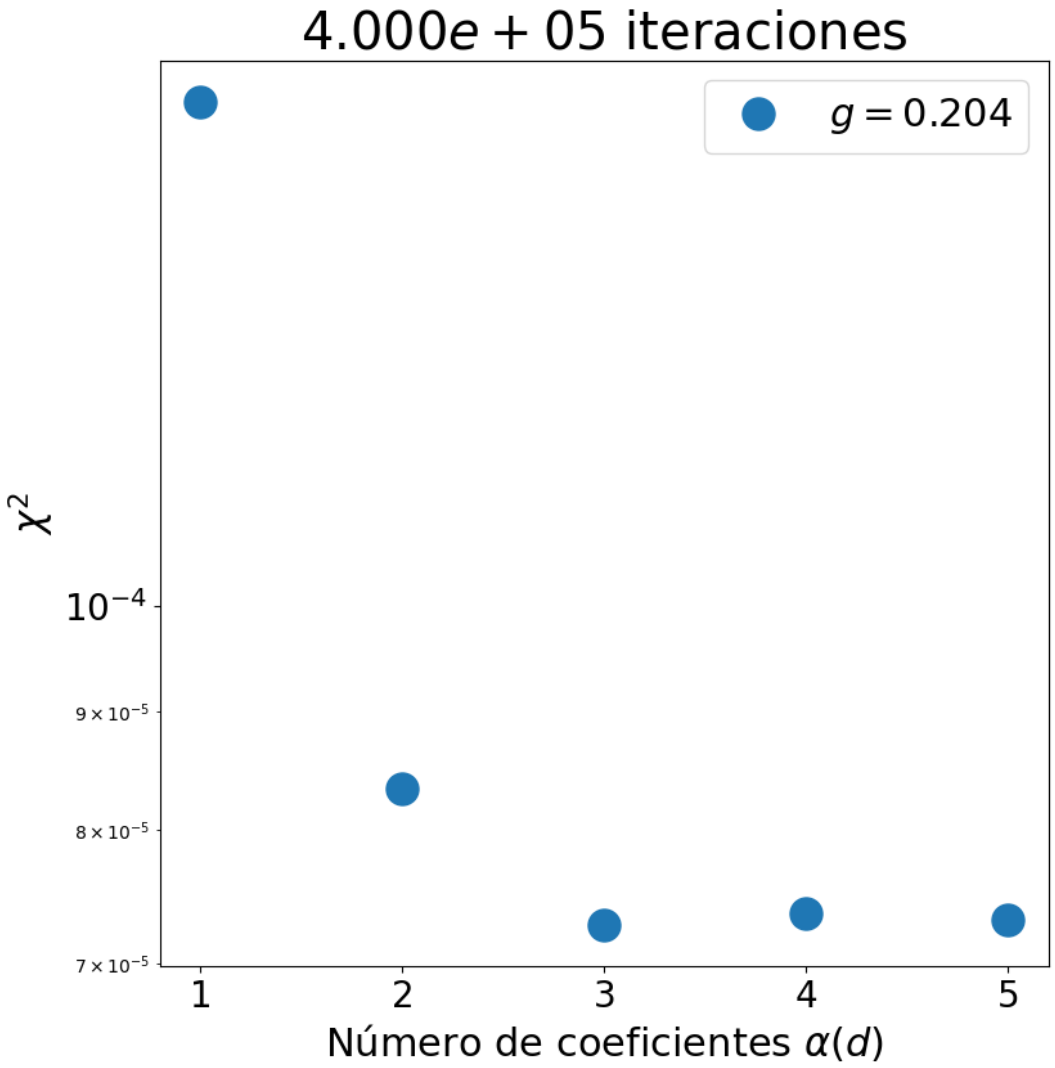


Figura 4.26: Efecto de la inclusión de coeficientes $\alpha(d)$ en el cálculo de la energía $\mathcal{H}(S_i)$ sobre la dispersión χ^2 en una rejilla "a grano grueso" construida a partir de un sistema CML de longitud $L = 32$, grano $G = 8$ y acoplamiento $g0.204$. La evolución temporal del sistema CG-CML fue de 4×10^5 iteraciones.

Capítulo 5

Conclusiones y perspectivas

> El presente capítulo resume los hallazgos de la tesis, ofrece explicaciones sobre las diferencias entre ésta y el artículo [5] y discute las posibles direcciones a seguir en futuras investigaciones sobre el tema.

Tras revisar la Sección 4, puedo afirmar que

- Las rejillas de mapeo acoplado descritas en el Capítulo 3 presentan ergodicidad, al menos para cantidades asociadas a sitios individuales. Esta propiedad no percibe cambios ante la modificación del acoplamiento g entre sitios, aunque su presencia puede ser afectada por la magnitud del periodo transitorio. En cambio, la condición de *self-averaging* no aparece en estos sistemas, puesto que las distribuciones de $u_{\vec{x}}^{t_0}$ en los sistemas de longitud $L = 64$ no coinciden con las distribuciones de $u_{\vec{x}_0}^{t}$ obtenidas para sistemas individuales o ensambles ni muestran una forma específica e independiente del acoplamiento o la duración del *transient*. Posiblemente las rejillas que estudiamos fueron demasiado pequeñas y contaban con un número de elementos insuficiente para exhibir *self-averaging*.

- El comportamiento de los momentos $\langle |u| \rangle_G$ corresponde al de una variable aleatoria normal o Gaussiana. Las simulaciones con longitudes superiores a $L = 24$ muestran el decaimiento propio de una variable de esta clase y son evidencia de la existencia de una escala en la que el caos del sistema CML funge el rol de un baño térmico.

- Las rejillas "a grano grueso" de 4×4 sitios derivadas de sistemas CML con tamaño de grano $G = 8$ presentan balance detallado, ya que los promedios de los factores $(\Delta/N)^2$ efectivamente decaen como N^{-1}. Las discrepancias entre el comportamiento esperado y los resultados de las simulaciones en sistemas CG-CML construidos a partir de rejillas con $G = 2$ son un orden de magnitud mayor que las diferencias para las pruebas en rejillas asociadas a $G = 8$, en conformidad con los límites planteados en [5]. Sin embargo, el peculiar *binning* que Egolf implementa en el cálculo de los promedios mencionados anteriormente produce resultados inesperados que sugieren que las desviaciones entre teoría y experimento en los sistemas examinados (sin importar el tamaño de grano) son similares y ambos tipos de rejillas "a grano grueso" cuentan con balance detallado. No es claro el propósito detrás de dicho procedimiento, por lo que determinamos que resulta más adecuado juzgar la condición de balance detallado a partir de los primeros resultados.

- Las distribuciones de probabilidad de los estados de las rejillas "a grano grueso" examinadas en este trabajo se ajustan razonablemente bien a distribuciones de Boltzmann,

considerando energías dadas por la Ecuación 3.6. Curiosamente, tal hecho es independiente de las dimensiones y el tamaño de grano de las rejillas "finas" usadas para elaborar los sistemas CG-CML, en contraste con los resultados obtenidos en las pruebas de balance detallado; la principal diferencia entre los resultados de las dos clases de rejillas radica en que el sistema CG-CML asociado a $G = 2$ tiene un coeficiente $\alpha(1)$ mayor a 1. El comportamiento de los coeficientes $\alpha(d)$ correspondientes a nuestras simulaciones es marcadamente distinto al de los valores reportados en [5], aunque los cambios de signo sí coinciden en el conjunto $\{\alpha(d)\}$ de la rejilla relacionada a un tamaño de grano $G = 8$.

En el futuro, resultaría conveniente repetir las pruebas de ergodicidad con valores adicionales para el acoplamiento g entre sitios de las rejillas de mapeo acoplado para comprobar que dichos sistemas exhiben tal propiedad en el rango destacado en [5] ($0.170 < g < 0.2054$). Analizar sistemas con mayor longitud o extender la duración de la evolución temporal (o el periodo transitorio) representan alternativas igualmente válidas. Además, sería interesante realizar más simulaciones para determinar el tamaño mínimo del sistema que asegura la presencia de *self-averaging* y examinar con mayor cuidado la relación entre tal propiedad y la magnitud de g.

Realizar simulaciones de balance detallado en rejillas de mapeo acoplado con diferentes longitudes L y tamaños de grano G podría confirmar que tal propiedad es observable en sistemas CG-CML asociados a $G \geq 8$ y proveer una justificación al proceso de *binning* que usa Egolf para estimar promedios. Consideramos que las rejillas "a grano grueso" deberían ser todas de 4×4 sitios, puesto que sistemas con más elementos cuentan con un total de estados órdenes de magnitud mayor, lo cual vuelve su estudio ineficiente; tan sólo un sistema CG-CML de 5×5 sitios tiene $2^{25} = 33554432$ estados.

Las nuevas simulaciones para la prueba de Gibbs deberían usar tiempos de evolución más largos para no volver a sobrestimar las probabilidades de los estados con energías mayores. Conforme al análisis presentado en el capítulo anterior, la duración óptima para la evolución temporal debería ser al menos del orden de 10^6 iteraciones. Tales medidas podrían verificar si los sistemas CG-CML con $G < 8$ efectivamente presentan una distribución de Gibbs. Adicionalmente, sería conveniente repetir el análisis de la distribución de Gibbs con diferentes números de coeficientes $\alpha(d)$ para dichos nuevos sistemas y así determinar el total de coeficientes que mejor reproducen tal distribución de probabilidad.

Confíamos en que seguir estas rutas de investigación podría sentar con mayor firmeza los resultados propuestos por Egolf y ofrecer un enfoque provechoso en el estudio de sistemas fuera de equilibrio. En particular, creemos que adaptar una rejilla de mapeo acoplado a un mercado financiero, un sistema conformado por agentes individuales en competencia entre sí bajo la influencia de un entorno inestable (como los que aparecen en [18], Capítulo 4), y aplicar la metodología propuesta por Egolf a su estudio sería una dirección prometedora en el análisis de tal clase de sistemas. Igualmente, sería beneficioso analizar los mismos sistemas aquí examinados incorporando acoplamientos entre sitios con mayor separación entre sí (como lo hace el código en el Apéndice $\tilde{\text{N}}$[1]) y observar el efecto que tales conexiones tienen sobre la presencia de balance detallado y distribuciones de Gibbs en rejillas "a grano grueso".

[1]Cabe resaltar que la inclusión de más acoplamientos resultaría en restricciones adicionales sobre sus valores g evitar que la regla de actualización arroje valores de u fuera del dominio del mapeo $\phi(u)$.

Apéndice A

Identidades básicas de probabilidad

En esta sección, presentamos definiciones e identidades básicas en el estudio de variables aleatorias según el desarrollo planteado en [12]. Adicionalmente, incluimos las formas que éstas adoptan al realizar cálculos numéricos.

A.1. Identidades fundamentales

A.1.1. Promedio

Sea U una variable aleatoria cualquiera. Su valor esperado o promedio $\langle U \rangle$ está definido como

$$\langle U \rangle = \sum_i u_i P(u_i) \tag{A.1}$$

donde u_i denota cada uno de los posibles valores que puede adquirir U y $P(u_i)$ representa su respectiva probabilidad. Considerando tal definición, sigue que

$$\langle aU \rangle = a \cdot \langle U \rangle \tag{A.2}$$

dada una constante a.

A.1.2. Varianza

La varianza $\mathrm{Var}\,[U]$ es el promedio del cuadrado de las diferencias entre la variable U y su promedio $\langle U \rangle$, es decir,

$$\begin{aligned} \mathrm{Var}\,[U] &= \left\langle (U - \langle U \rangle)^2 \right\rangle \\ &= \sum_i \left(u_i - \langle U \rangle\right)^2 P(u_i) \end{aligned} \tag{A.3}$$

Tomando en cuenta la Ecuación A.2 y la definición de la varianza, resulta que $\mathrm{Var}\,[aU]$ (siendo a una constante) es igual a

$$\mathrm{Var}\,[aU] = a^2 \cdot \mathrm{Var}\,[U] \tag{A.4}$$

La Ecuación A.3 implica también que

$$\mathrm{Var}\left[U\right] = \left\langle U^2 \right\rangle - \left\langle U \right\rangle^2 \tag{A.5}$$

A.1.3. Variables independientes

Dos variables aleatorias X y Y son estadísticamente independientes entre sí al satisfacer que

$$P\left(x_i \;\&\; y_j\right) = P\left(x_i\right) \cdot P\left(y_j\right) \tag{A.6}$$

El promedio del producto entre dichas variables es tal que

$$\begin{aligned} \left\langle XY \right\rangle &= \sum_i \sum_j x_i y_i P\left(x_i \;\&\; y_i\right) \\ &= \sum_i x_i P\left(x_i\right) \sum_j y_j P\left(y_j\right) \\ &= \left\langle X \right\rangle \cdot \left\langle Y \right\rangle \end{aligned} \tag{A.7}$$

A.1.4. Promedio y varianza de la suma de variables aleatorias

Partiendo de la definición del promedio (Ecuación A.1), resulta inmediato demostrar que el promedio de la suma de las variables aleatorias X y Y es equivalente a la suma de sus respectivos promedios, de modo tal que

$$\left\langle X + Y \right\rangle = \left\langle X \right\rangle + \left\langle Y \right\rangle \tag{A.8}$$

La condición anterior puede extenderse a la suma de n variables aleatorias. Dicho resultado queda sucintamente planteado como

$$\left\langle \sum_j^n U_j \right\rangle = \sum_j^n \left\langle U_j \right\rangle \tag{A.9}$$

En cambio, la varianza de la suma de las variables X y Y solamente puede expresarse como la suma de sus varianzas si las variables son independientes; bajo tal condición, se tiene que

$$\begin{aligned} \mathrm{Var}\left[X + Y\right] &= \left\langle \left[(X + Y) - \left\langle X + Y \right\rangle\right]^2 \right\rangle && \text{[A.3]} \\ &= \left\langle \left(X - \left\langle X \right\rangle\right)^2 \right\rangle + \left\langle \left(Y - \left\langle Y \right\rangle\right)^2 \right\rangle + 2\left[\left\langle XY \right\rangle - \left\langle X \right\rangle \left\langle Y \right\rangle\right] && \text{[A.2, A.9]} \\ &= \mathrm{Var}\left[X\right] + \mathrm{Var}\left[Y\right] && \text{[A.7]} \end{aligned} \tag{A.10}$$

Al considerar n variables independientes, la varianza de su suma es exactamente igual a la suma de sus varianzas correspondientes:

$$\mathrm{Var}\left[\sum_j^n U_j\right] = \sum_j^n \mathrm{Var}\left[U_j\right] \tag{A.11}$$

A.2. Expresiones numéricas

Al contar con un conjunto de M datos o mediciones, el promedio y la varianza de una variable aleatoria U son calculados respectivamente como

$$\text{Prom}\,[U] = \overline{U} = \frac{1}{M}\sum_{i}^{M} u_i \tag{A.12}$$

$$\text{Var}\,[U] = \frac{1}{M}\sum_{i}^{M} (u_i - \overline{U})^2 \tag{A.13}$$

Cabe resaltar que la expresión de la varianza anterior es válida si los M valores de U conforman la población completa de tal variable. En caso de que este conjunto sea únicamente una muestra, el denominador en la Ecuación A.13 es $M - 1$.

Las definiciones anteriores implican que las Ecuaciones A.2 y A.4 siguen siendo verdaderas, tras adaptarlas a cálculos numéricos.

Apéndice B

Momentos centrales de una variable aleatoria con distribución Gaussiana

La función $\exp\{-ax^2\}$ (donde a es un número real positivo) recibe el nombre de Guassiana. Como su antiderivada no cuenta con una forma analítica, la integral de esta función suele calcularse numéricamente. Sin embargo, la integral de la función Gaussiana sobre todos los números reales sí tiene un valor definido:

$$\int_{-\infty}^{\infty} \exp\{-ax^2\}\ dx = \sqrt{\frac{\pi}{a}} \tag{B.1}$$

Conforme al desarrollo presentado en [17], pp. 384-386, la demostración de la Ecuación B.1 considera

$$I = \int_{-\infty}^{\infty} \exp\{-ax^2\}\ dx \tag{B.2}$$

Entonces,

$$\begin{aligned} I^2 &= \left(\int_{-\infty}^{\infty} \exp\{-ax^2\}\ dx\right)^2 \\ &= \left(\int_{-\infty}^{\infty} \exp\{-ax^2\}\ dx\right)\left(\int_{-\infty}^{\infty} \exp\{-ay^2\}\ dy\right) \\ &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \exp\{-a(x^2+y^2)\}\ dx\ dy \end{aligned} \tag{B.3}$$

Con un cambio a coordenadas polares, sigue que

$$
\begin{aligned}
I^2 &= \int_0^{\infty} \int_0^{2\pi} \exp\{-ar^2\}\ r\ dr\ d\theta \\
&= 2\pi \int_0^{\infty} \exp\{-ar^2\}\ r\ dr \\
&= 2\pi \left[-\frac{\exp\{-ar^2\}}{2a} \right] \Bigg|_0^{\infty} \\
&= \frac{\pi}{a}
\end{aligned} \tag{B.4}
$$

La Ecuación B.1 resulta de gran utilidad al calcular integrales de la forma $\int_{-\infty}^{\infty} x^{2n}\ \exp\{-ax^2\}\ dx$, donde n es un número natural.

Para $n = 1$, se tiene que

$$
\begin{aligned}
\int_{-\infty}^{\infty} x^2\ \exp\{-ax^2\}\ dx &= \int_{-\infty}^{\infty} \frac{d}{da} \left[-\exp\{-ax^2\} \right]\ dx = -\frac{d}{da} \left[\int_{-\infty}^{\infty} \exp\{-ax^2\}\ dx \right] \\
&= -\frac{d}{da} \left[\sqrt{\frac{\pi}{a}}\ \right] \\
&= \frac{1}{2} \sqrt{\frac{\pi}{a^3}}
\end{aligned} \tag{B.5}
$$

Análogamente, si $n = 2$ o $n = 3$, resulta que

$$
\begin{aligned}
\int_{-\infty}^{\infty} x^4\ \exp\{-ax^2\}\ dx &= \int_{-\infty}^{\infty} \frac{d}{da} \left[-x^2 \exp\{-ax^2\} \right]\ dx = -\frac{d}{da} \left[\int_{-\infty}^{\infty} x^2 \exp\{-ax^2\}\ dx \right] \\
&= -\frac{d}{da} \left[\frac{1}{2} \sqrt{\frac{\pi}{a^3}}\ \right] \\
&= \frac{3}{4} \sqrt{\frac{\pi}{a^5}}
\end{aligned} \tag{B.6}
$$

$$
\begin{aligned}
\int_{-\infty}^{\infty} x^6\ \exp\{-ax^2\}\ dx &= \int_{-\infty}^{\infty} \frac{d}{da} \left[-x^4 \exp\{-ax^2\} \right]\ dx = -\frac{d}{da} \left[\int_{-\infty}^{\infty} x^4 \exp\{-ax^2\}\ dx \right] \\
&= -\frac{d}{da} \left[\sqrt{\frac{3}{4} \frac{\pi}{a^5}}\ \right] \\
&= \frac{15}{8} \sqrt{\frac{\pi}{a^7}}
\end{aligned} \tag{B.7}
$$

En contraste, las integrales $\int_{-\infty}^{\infty} x^{2n-1}\ \exp\{-ax^2\}\ dx$ son todas equivalentes a cero. La función x^{2n-1} es impar y $\exp\{-ax^2\}$, par. Por consiguiente, $x^{2n-1}\ \exp\{-ax^2\}$ es una función impar. Como cualquier integral de función impar con límites que difieren únicamente por un signo es nula, sigue que

$$\int_{-\infty}^{\infty} x^{2n-1}\ \exp\{-ax^2\}\ dx = 0 \quad \forall n \in \mathbb{N} \tag{B.8}$$

En particular,

$$\int_{-\infty}^{\infty} x\ \exp\{-ax^2\}\ dx = \left[-\frac{1}{2a}\exp\{-ax^2\}\right]\Bigg|_{-\infty}^{\infty} \tag{B.9}$$

$$\begin{aligned}\int_{-\infty}^{\infty} x^3\ \exp\{-ax^2\}\ dx &= \int_{-\infty}^{\infty} x^2\ x\exp\{-ax^2\}\ dx \\ &= \left[-\frac{x^2}{2a}\exp\{-ax^2\}\right]\Bigg|_{-\infty}^{\infty} - \int_{-\infty}^{\infty} -\frac{x}{a}\ \exp\{-ax^2\}\ dx \\ &= 0\end{aligned} \tag{B.10}$$

Conforme a [12], p. 25, una variable aleatoria normal o Gaussiana z con promedio μ y varianza σ^2 cuenta con función de densidad de probabilidad $p(z)$ dada por

$$p(z) = \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left\{-\frac{(z-\mu)^2}{2\sigma^2}\right\} \tag{B.11}$$

Por consiguiente, el momento central n $\langle(z-\mu)^n\rangle$ está definido como

$$\langle(z-\mu)^n\rangle = \int_{-\infty}^{\infty}(z-\mu)^n\ \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left\{-\frac{(z-\mu)^2}{2\sigma^2}\right\}\ dz \tag{B.12}$$

Considerando $\tilde{z} = z - \mu$ y $\alpha = \frac{1}{2\sigma^2}$, sigue que

$$\langle(z-\mu)^n\rangle = \frac{1}{\sqrt{2\pi\sigma^2}}\int_{-\infty}^{\infty}\tilde{z}^n\ \exp\{-\alpha\tilde{z}^2\}\ d\tilde{z} \tag{B.13}$$

Entonces, las Ecuaciones B.5, B.6 y B.7 implican que

$$\langle(z-\mu)^2\rangle = \frac{1}{\sqrt{2\pi\sigma^2}}\cdot\left(\frac{1}{2}\sqrt{\frac{\pi}{\alpha^3}}\right) = \sigma^2 \tag{B.14}$$

$$\langle(z-\mu)^4\rangle = \frac{1}{\sqrt{2\pi\sigma^2}}\cdot\left(\frac{3}{4}\sqrt{\frac{\pi}{\alpha^5}}\right) = 3\sigma^4 \tag{B.15}$$

$$\langle(z-\mu)^6\rangle = \frac{1}{\sqrt{2\pi\sigma^2}}\cdot\left(\frac{15}{8}\sqrt{\frac{\pi}{\alpha^7}}\right) = 15\sigma^6 \tag{B.16}$$

También sería correcto afirmar que

$$\left(\frac{4}{3}\langle(z-\mu)^4\rangle\right)^{\frac{1}{2}} = \left(\frac{4}{3}\cdot 3\sigma^4\right)^{\frac{1}{2}} = 2\sigma^2 \tag{B.17}$$

$$\left(\langle(z-\mu)^6\rangle\right) = \frac{1}{\sqrt{2\pi\sigma^2}}\cdot\left(\frac{15}{8}\sqrt{\frac{\pi}{\alpha^7}}\right) = 15\sigma^6 \tag{B.18}$$

Apéndice C

Demostración de identidad de balance detallado en rejilla de mapeo acoplado a grano grueso

Sea $z_k^{(i,j)}$ una variable aleatoria definida como

$$z_k^{(i,j)} = \begin{cases} z_{k,+1}^{(i,j)} = +1 & P\left(z_{k,+1}^{(i,j)}\right) = \frac{1}{2} \\ z_{k,-1}^{(i,j)} = -1 & P\left(z_{k,-1}^{(i,j)}\right) = \frac{1}{2} \end{cases} \tag{C.1}$$

Esta variable sirve para marcar el sentido de la transición k entre dos estados (i, j) que presenta una rejilla de mapeo acoplado a grano grueso dotada de balance detallado durante su evolución, es decir, indica si el sistema experimenta una transición $i \to j$ o una transición $j \to i$. Dadas $N^{(i,j)}$ transiciones entre el par de estados (i, j), la diferencia $\Delta^{(i,j)}$ puede expresarse como

$$\Delta^{(i,j)} = \sum_{k=1}^{N^{(i,j)}} z_k^{(i,j)} \tag{C.2}$$

Conforme a las Ecuaciones A.1 y A.5, la definición de $z_k^{(i,j)}$ implica que

$$\begin{aligned} \left\langle z_k^{(i,j)} \right\rangle &= \sum_x z_{k,x}^{(i,j)} P\left(z_{k,x}^{(i,j)}\right) \\ &= z_{k,+1}^{(i,j)} P\left(z_{k,+1}^{(i,j)}\right) + z_{k,-1}^{(i,j)} P\left(z_{k,-1}^{(i,j)}\right) \\ &= (1)\left(\frac{1}{2}\right) + (-1)\left(\frac{1}{2}\right) \\ &= 0 \end{aligned} \tag{C.3}$$

$$\begin{aligned}\left\langle \left(z_k^{(i,j)}\right)^2 \right\rangle &= \sum_x \left(z_{k,x}^{(i,j)}\right)^2 P\left(z_{k,x}^{(i,j)}\right) \\ &= \left(z_{k,+1}^{(i,j)}\right)^2 P\left(z_{k,+1}^{(i,j)}\right) + \left(z_{k,-1}^{(i,j)}\right)^2 P\left(z_{k,-1}^{(i,j)}\right) \\ &= (1)^2\left(\frac{1}{2}\right) + (-1)^2\left(\frac{1}{2}\right) \\ &= 1 \end{aligned} \tag{C.4}$$

$$\begin{aligned}\mathrm{Var}\left[z_k^{(i,j)}\right] &= \left\langle \left(z_k^{(i,j)}\right)^2 \right\rangle - \left\langle z_k^{(i,j)} \right\rangle^2 \\ &= 1 \end{aligned} \tag{C.5}$$

A continuación, propongo dos formas distintas de estimar el factor $\left\langle \left(\frac{\Delta^{(i,j)}}{N}\right)^2 \right\rangle$ asumiendo que el sistema CG-CML presenta balance detallado y la naturaleza de sus transiciones está debidamente representada en la variable $z_k^{(i,j)}$.

C.1. Demostración 1

Estableciendo que $N^{(i,j)} = N$, la Ecuación A.2 asegura que

$$\left\langle \left(\frac{\Delta^{(i,j)}}{N}\right)^2 \right\rangle = \frac{1}{N^2}\left\langle \left(\Delta^{(i,j)}\right)^2 \right\rangle \tag{C.6}$$

Conforme a la definición de la diferencia $\Delta^{(i,j)}$ (Ecuación C.2), sigue que

$$\begin{aligned}\left\langle \left(\Delta^{(i,j)}\right)^2 \right\rangle &= \left\langle \left(\sum_k^N z_k^{(i,j)}\right)^2 \right\rangle = \left\langle \left(z_1^{(i,j)} + \cdots + z_N^{(i,j)}\right)\left(z_1^{(i,j)} + \cdots + z_N^{(i,j)}\right) \right\rangle \\ &= \left\langle \sum_k^N \left(z_k^{(i,j)}\right)^2 + \sum_l^N \sum_{m(m\neq l)}^N z_l^{(i,j)} z_m^{(i,j)} \right\rangle \end{aligned} \tag{C.7}$$

Considerando que cada transición (y su correspondiente $z_k^{(i,j)}$) es independiente de las demás, las Ecuaciones A.7 y A.9 garantizan que

$$\begin{aligned}\left\langle \left(\Delta^{(i,j)}\right)^2 \right\rangle &= \left\langle \sum_k^N \left(z_k^{(i,j)}\right)^2 \right\rangle + \left\langle \sum_l^N \sum_{m(m\neq l)}^N z_l^{(i,j)} z_m^{(i,j)} \right\rangle \\ &= \sum_k^N \left\langle \left(z_k^{(i,j)}\right)^2 \right\rangle + \sum_l^N \sum_{m(m\neq l)}^N \left\langle z_l^{(i,j)} z_m^{(i,j)} \right\rangle \\ &= \sum_k^N \left\langle \left(z_k^{(i,j)}\right)^2 \right\rangle + \sum_l^N \sum_{m(m\neq l)}^N \left\langle z_l^{(i,j)} \right\rangle \left\langle z_m^{(i,j)} \right\rangle \end{aligned} \tag{C.8}$$

Con las Ecuaciones C.3 y C.4, es posible afirmar que

$$\left\langle \left(\Delta^{(i,j)}\right)^2 \right\rangle = \sum_k^N \langle 1 \rangle + \sum_l^N \sum_{m(m\neq l)}^N (0)\cdot(0) = N \tag{C.9}$$

Así, es correcto afirmar que

$$\left\langle \left(\frac{\Delta^{(i,j)}}{N}\right)^2 \right\rangle = \frac{1}{N^2}\cdot N = N^{-1} \tag{C.10}$$

C.2. Demostración 2

Con la Ecuación A.5, el factor $\left\langle \left(\frac{\Delta^{(i,j)}}{N}\right)^2 \right\rangle$ puede expresarse (bajo la misma constricción usada en la demostración anterior) como

$$\begin{aligned}\left\langle \left(\frac{\Delta^{(i,j)}}{N}\right)^2 \right\rangle &= \frac{1}{N^2}\left\langle \left(\Delta^{(i,j)}\right)^2 \right\rangle \\ &= \frac{1}{N^2}\left[\mathrm{Var}\left[\Delta^{(i,j)}\right] + \left\langle \Delta^{(i,j)} \right\rangle^2\right] \end{aligned} \tag{C.11}$$

Por un lado, la Ecuación A.11, junto con las definiciones de $\Delta^{(i,j)}$ (Ecuación C.2) y la varianza de $z_k^{(i,j)}$ (Ecuación C.5), implica que

$$\begin{aligned}\mathrm{Var}\left[\Delta^{(i,j)}\right] &= \mathrm{Var}\left[\sum_{k=1}^N z_k^{(i,j)}\right] \\ &= \sum_{k=1}^N \mathrm{Var}\left[z_k^{(i,j)}\right] \\ &= \sum_{k=1}^N 1 = N \end{aligned} \tag{C.12}$$

Por el otro, la Ecuación A.9 y el valor del promedio de $z_k^{(i,j)}$ (Ecuación C.3) sugieren que

$$\begin{aligned}\left\langle \Delta^{(i,j)} \right\rangle^2 &= \left\langle \sum_{k=1}^{N} z_k^{(i,j)} \right\rangle^2 \\ &= \left(\sum_{k=1}^{N} \left\langle z_k^{(i,j)} \right\rangle \right)^2 \\ &= \left(\sum_{k=1}^{N} (0) \right)^2 = 0 \end{aligned} \tag{C.13}$$

Por lo tanto, resulta que

$$\left\langle \left(\frac{\Delta^{(i,j)}}{N} \right)^2 \right\rangle = \frac{1}{N^2} \left[N + 0 \right] = N^{-1} \tag{C.14}$$

Apéndice D

Código para prueba de ergodicidad en rejilla de mapeo acoplado

```
import sys
import random
import numpy as np
import networkx as nx
import matplotlib.pyplot as plt

##############################################################################
##############################################################################

### CÁLCULO DE DISTANCIAS ENTRE DOS ELEMENTOS DE UNA REJILLA CUADRADA
    BIDIMENSIONAL DOTADA DE CONDICIONTES DE FRONTERA PERIÓDICAS
def calc_dist(loc1, loc2, longitud):
  distx = loc1[0] - loc2[0]
  disty = loc1[1] - loc2[1]
  dist1 = (distx ** 2 + disty ** 2) ** (1 / 2)
  dist2 = ((distx - longitud) ** 2 + disty ** 2) ** (1 / 2)
  dist3 = ((distx + longitud) ** 2 + disty ** 2) ** (1 / 2)
  dist4 = (distx ** 2 + (disty - longitud) ** 2) ** (1 / 2)
  dist5 = (distx ** 2 + (disty + longitud) ** 2) ** (1 / 2)
  dist6 = ((distx + longitud) ** 2 + (disty + longitud) ** 2) ** (1 / 2)
  dist7 = ((distx - longitud) ** 2 + (disty + longitud) ** 2) ** (1 / 2)
  dist8 = ((distx - longitud) ** 2 + (disty - longitud) ** 2) ** (1 / 2)
  dist9 = ((distx + longitud) ** 2 + (disty - longitud) ** 2) ** (1 / 2)
  lista_distancias = [dist1, dist2, dist3, dist4, dist5, dist6, dist7, dist8
    , dist9]
  distancia = min(lista_distancias)
  return distancia

### INICIACIÓN DE GRÁFICA PARA SISTEMA CML
def ini_graf(longitud):
  grafica = nx.Graph()
  # Definición de nodos
  num_sitios = longitud ** 2
  for sitio in range(num_sitios):
    grafica.add_node(sitio, u = random.uniform(-1, 1))
  # Definición de bordes
  lista_edges = []
  for sitio1 in range(num_sitios):
    for sitio2 in range(num_sitios):
      if sitio2 == sitio1:
```

```
39          continue
40        coord1 = (sitio1 % longitud, sitio1 // longitud)
41        coord2 = (sitio2 % longitud, sitio2 // longitud)
42        distancia = calc_dist(coord1, coord2, longitud)
43        lista_edges.append((sitio1, sitio2, {'path_distance': distancia}))
44    grafica.add_edges_from(lista_edges)
45    return grafica
46
47 ### CÁLCULO DE PRIMEROS VECINOS DE UN NODO EN UNA GRÁFICA CON BORDES DOTADOS
       DE ATRIBUTOS EQUIVALENTES A LAS DISTANCIAS ENTRE LOS SITIOS QUE LOS
      DEFINEN
48 def primeros_vecinos(grafica, nodo):
49    # Lista de posibles distancias
50    lista_dist = []
51    for vec1, datos1 in grafica.adj[nodo].items():
52      for keys1, dists1 in datos1.items():
53        lista_dist.append(dists1)
54    min_dist = min(lista_dist)
55    # Lista de sitios a distancia mínima (primeros vecinos)
56    lista_1v = []
57    for vec2, datos2 in grafica.adj[nodo].items():
58      for keys2, dists2 in datos2.items():
59        if dists2 == min_dist:
60          lista_1v.append(vec2)
61    return lista_1v
62
63 ### MAPEO PHI
64 def phi(valor_t):
65    if -1 <= valor_t < -1 / 3.:
66      valor_tt = (-3 * valor_t) - 2
67    elif -1 / 3. <= valor_t < 1 / 3.:
68      valor_tt = 3 * valor_t
69    elif 1 / 3. <= valor_t <= 1:
70      valor_tt = (-3 * valor_t) + 2
71    return valor_tt
72
73 ### EVOLUCIÓN TEMPORAL DE LA GRÁFICA
74 def ev_temp(num_iter, trans, x0, g_acople, grafica, lista_1vecinos, guardar)
      :
75    print('INICIO DE EVOLUCION TEMPORAL')
76    lista_sitios = list(grafica.nodes())
77    tenth_progress = int(num_iter / 10)
78    # Guardar evolución de 'u'
79    if guardar == True:
80      arr_promtemp = np.zeros((num_iter - trans))
81      for iteracion1 in range(num_iter):
82        if iteracion1 % tenth_progress == 0:
83          print('*')
84        # Gráfica auxiliar con valores de 'u' de siguiente iteración
85        grafica_holder1 = nx.Graph()
86        for sitio1a in lista_sitios:
87          grafica_holder1.add_node(sitio1a, u = 0)
88          sum1vec1 = 0
89          for vecino1 in lista_1vecinos[sitio1a][1]:
90            dif_u1 = phi(grafica.nodes[vecino1]['u']) - phi(grafica.nodes[
      sitio1a]['u'])
91            sum1vec1 = sum1vec1 + dif_u1
```

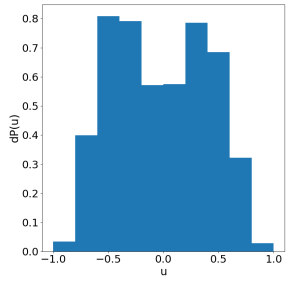
```
92          grafica_holder1.nodes[sitio1a]['u'] = phi(grafica.nodes[sitio1a]['u'
    ]) + g_acople * sum1vec1
93        # Actualización de gráfica "original"
94        for sitio1b in lista_sitios:
95          grafica.nodes[sitio1b]['u'] = grafica_holder1.nodes[sitio1b]['u']
96        if iteracion1 <= trans - 1:
97          continue
98        arr_promtemp[iteracion1 - trans] = grafica.nodes[x0]['u']
99      print('FIN DE EVOLUCION TEMPORAL')
100     return grafica, arr_promtemp
101   # Sin guardar evolución de 'u'
102   else:
103     for iteracion2 in range(num_iter):
104       if iteracion2 % tenth_progress == 0:
105         print('*')
106       # Gráfica auxiliar
107       grafica_holder2 = nx.Graph()
108       for sitio2a in lista_sitios:
109         grafica_holder2.add_node(sitio2a, u = 0)
110         sum1vec2 = 0
111         for vecino2 in lista_1vecinos[sitio2a][1]:
112           dif_u2 = phi(grafica.nodes[vecino2]['u']) - phi(grafica.nodes[
    sitio2a]['u'])
113           sum1vec2 = sum1vec2 + dif_u2
114         grafica_holder2.nodes[sitio2a]['u'] = phi(grafica.nodes[sitio2a]['u'
    ]) + g_acople * sum1vec2
115       # Actualización de gráfica
116       for sitio2b in lista_sitios:
117         grafica.nodes[sitio2b]['u'] = grafica_holder2.nodes[sitio2b]['u']
118     print('FIN DE EVOLUCION TEMPORAL')
119     return grafica
120
121 ##############################################################################
122 ##############################################################################
123
124 ### DEFINICIÓN DE PARÁMETROS DE SIMULACIÓN
125 # Selección aleatoria de semilla
126 s = random.randrange(sys.maxsize)
127 # Definición de otros parámetros
128 L = int(input('Ingresa la longitud de la rejilla CML (entero): '))
129 g = float(input('Ingresa la constante de acoplamiento (real positivo con
    tres decimales): '))
130 N_iter = int(input('Ingresa el total de iteraciones (entero): '))
131 transient = int(input('Ingresa el valor de transient (entero): '))
132 site_x0 =  int(random.randrange(0,int(L**2),1))
133 N_ensembles = int(input('Ingresa el total de sistemas que conforman el
    ensamble (entero): '))
134
135 ### GENERACIÓN DE GRÁFICA Y LISTA CON PRIMEROS VECINOS
136 # Iniciación de generador de números aleatorios
137 random.seed(s)
138 # Definición de gráfica y lista con primeros vecinos
139 lattice = ini_graf(L)
140 list_1neighbors = []
141 for site in range(L**2):
142   list1v = primeros_vecinos(lattice, site)
143   list_1neighbors.append((site, list1v))
144
```

```
145 ### DISTRIBUCIÓN DE VARIABLE 'U' EN UN SITIO PARTICULAR TRAS MÚLTIPLES
        ITERACIONES, CONSIDERANDO UN SISTEMA
146 safe_timeavg = True
147 lattice, arr_timeavg = ev_temp(N_iter, transient, site_x0, g, lattice,
        list_1neighbors, safe_timeavg)
148
149 fname_timeavg = 'tesis_Egolf_ergodicidad_L%(length)i_g%(coupling).3f_Niter%(
        iterations).3e_trans%(trans).3e_seed%(seed)i_site%(site)i_timeavg.txt'
150 dict_fname_timeavg = {'length': L, 'coupling': g, 'iterations': N_iter, '
        trans': transient, 'site': site_x0, 'seed': s}
151 np.savetxt(fname_timeavg % dict_fname_timeavg, arr_timeavg)
152
153 print('Evolución de sistema completada')
154
155 # Distribución de variable 'u' en un sitio particular tras múltiples
        iteraciones, considerando un ensamble
156 safe_ensembleavg = True
157 arr_ensembleavg = np.zeros((N_ensembles, (N_iter - transient)))
158 for sys in range(N_ensembles):
159   lattice = ini_graf(L)
160   lattice, arr_ensemble_holder = ev_temp(N_iter, transient, site_x0, g,
        lattice, list_1neighbors, safe_ensembleavg)
161   arr_ensembleavg[sys] = arr_ensemble_holder
162 arr_ensembleavg = arr_ensembleavg.flatten()
163
164 fname_ensavg = 'tesis_Egolf_ergodicidad_L%(length)i_g%(coupling).3f_Niter%(
        iterations).3e_trans%(trans).3e_seed%(seed)i_site%(site)i_Nens%(ens)
        i_ensavg.txt'
165 dict_fname_ensavg = {'length': L, 'coupling': g, 'iterations': N_iter, '
        trans': transient, 'site': site_x0, 'ens': N_ensembles, 'seed': s}
166 np.savetxt(fname_ensavg % dict_fname_ensavg, arr_ensembleavg)
167
168 print('Evolución de ensamble completada')
169
170 # Iniciación de rejilla CML para prueba de self-averaging
171 L2 = int(2*L)
172 lattice2 = ini_graf(L2)
173 list_1neighbors2 = []
174 for site2 in range(L2**2):
175   list1v2 = primeros_vecinos(lattice2, site2)
176   list_1neighbors2.append((site2, list1v2))
177
178 # Distribución de variable 'u' en un sistema a un tiempo fijo
179 safe_selfavg = False
180 lattice2 = ev_temp(N_iter, transient, site_x0, g, lattice2, list_1neighbors2
        , safe_selfavg)
181 arr_selfavg = np.zeros(L2**2)
182 for site in range(L2**2):
183   arr_selfavg[site] = lattice2.nodes[site]['u']
184
185 fname_selfavg = 'tesis_Egolf_ergodicidad_L%(length)i_g%(coupling).3f_Niter%(
        iterations).3e_trans%(trans).3e_seed%(seed)i_selfavg.txt'
186 dict_fname_selfavg = {'length': L, 'coupling': g, 'iterations': N_iter, '
        trans': transient, 'seed': s}
187 np.savetxt(fname_selfavg % dict_fname_selfavg, arr_selfavg)
188
189 print('Prueba de self-averaging completada')
190
```

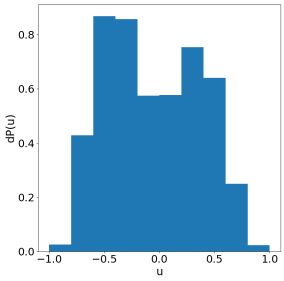
```
191 # Gráfica con resultados de ergodicidad y self-averaging
192 plt.figure(1)
193 fig1, (ax1A, ax1B, ax1C) = plt.subplots(nrows = 1, ncols = 3, figsize =
    (30,10))
194 plt.tight_layout(pad=4, h_pad=4, w_pad=6)
195
196 hist_time, bins_time = np.histogram(arr_timeavg, range = (-1, 1))
197 hist_ens, bins_ens = np.histogram(arr_ensembleavg, range = (-1, 1))
198 hist_self, bins_self = np.histogram(arr_selfavg, range = (-1,1))
199
200 ax1A.hist(bins_time[:-1], bins_time, weights = hist_time, density = True)
201 ax1A.set_title('Distribución de ' + r'$u_{\vec{x}_{0}}^{t}$' + ' con un
    sistema de %(lo)i x %(lo)i sitios ' % {'lo': L} + r'$(\vec{x}_{0} = %(
    siteref)i) $' % {'siteref': site_x0}, size=16)
202 ax1A.set_ylabel('dP(u)', size=15)
203 ax1A.set_xlabel('u', size=15)
204 for tick in ax1A.xaxis.get_major_ticks():
205   tick.label.set_fontsize(14)
206 for tick in ax1A.yaxis.get_major_ticks():
207   tick.label.set_fontsize(14)
208
209 ax1B.hist(bins_ens[:-1], bins_ens, weights = hist_ens, density = True)
210 ax1B.set_title('Distribución de ' + r'$u_{\vec{x}_{0}}^{t}$' + ' con un
    ensamble de %(Nens)i sistemas de %(lo)i x %(lo)i sitios' % {'Nens':
    N_ensembles, 'lo': L}, size=16)
211 ax1B.set_ylabel('dP(u)', size=15)
212 ax1B.set_xlabel('u', size=15)
213 for tick in ax1B.xaxis.get_major_ticks():
214   tick.label.set_fontsize(14)
215 for tick in ax1B.yaxis.get_major_ticks():
216   tick.label.set_fontsize(14)
217
218 ax1C.hist(bins_self[:-1], bins_self, weights = hist_self, density = True)
219 ax1C.set_title('Distribución de ' + r'$u_{\vec{x}}^{t_{0}}$' + ' sobre un
    sistema de %(lo)i x %(lo)i sitios ' % {'lo': L2} + r'$(t_{0} = %(tiempo)
    .2e) $' % {'tiempo': N_iter}, size=16)
220 ax1C.set_ylabel('dP(u)', size=15)
221 ax1C.set_xlabel('u', size=15)
222 for tick in ax1C.xaxis.get_major_ticks():
223   tick.label.set_fontsize(14)
224 for tick in ax1C.yaxis.get_major_ticks():
225   tick.label.set_fontsize(14)
226
227 imgname = 'tesis_Egolf_ergodicidad_L%(length)i_g%(coupling).3f_Niter%(
    iterations).3e_trans%(trans).3e_seed%(seed)i_site%(site)i_Nens%(ens)
    i_Graph.png'
228 dict_imgname = {'length': L, 'coupling': g, 'iterations': N_iter, 'trans':
    transient, 'site': site_x0, 'ens': N_ensembles, 'seed': s}
229 plt.savefig(imgname % dict_imgname)
230
231 print('Programa concluido')
```

Apéndice E

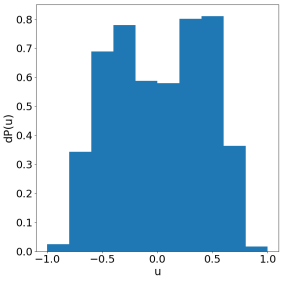
Gráficas de simulaciones para prueba de *self-averaging*



(a) Distribución de $u_{\vec{x}}^{t_0}$
($s = 4452967705924205789$)

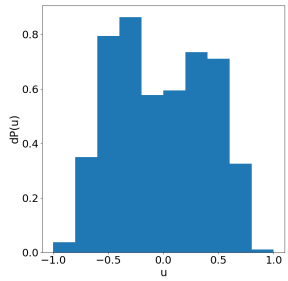


(b) Distribución de $u_{\vec{x}}^{t_0}$
($s = 5499070122743521182$)

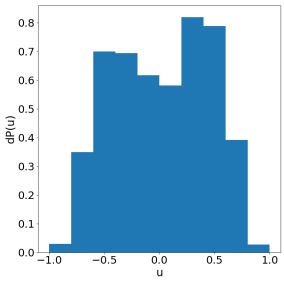


(c) Distribución de $u_{\vec{x}}^{t_0}$
($s = 7092030987844978395$)

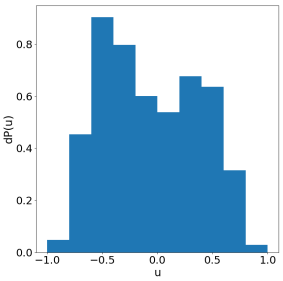
Figura E.1: Distribuciones de u para rejillas de mapeo acoplado de 64×64 sitios y acoplamiento $g = 0.195$ para diferentes semillas s. Las mediciones de `'u'` son efectuadas sobre el sistema completo al final de una evolución temporal de 2×10^5 iteraciones y un periodo transitorio de 10^3.



(a) Distribución de $u_{\vec{x}}^{t_0}$
($s = 1131207041169938644$)

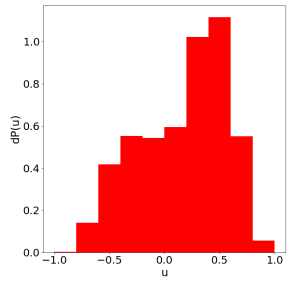


(b) Distribución de $u_{\vec{x}}^{t_0}$
($s = 5026122200688730582$)

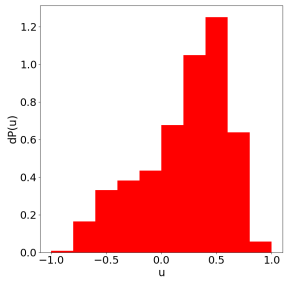


(c) Distribución de $u_{\vec{x}}^{t_0}$
($s = 7880188310502227256$)

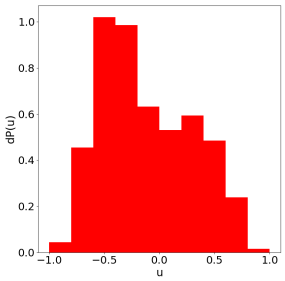
Figura E.2: Distribuciones de u para rejillas de mapeo acoplado de 64×64 sitios y acoplamiento $g = 0.195$ para diferentes semillas s. Las mediciones de `'u'` son efectuadas sobre el sistema completo al final de una evolución temporal de 2×10^5 iteraciones y un periodo transitorio de 10^4.



(a) Distribución de $u_{\vec{x}}^{t_0}$ ($s = 4536755601090925278$)

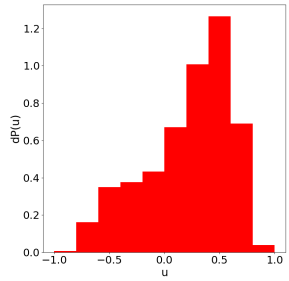


(b) Distribución de $u_{\vec{x}}^{t_0}$ ($s = 5535795909422011662$)

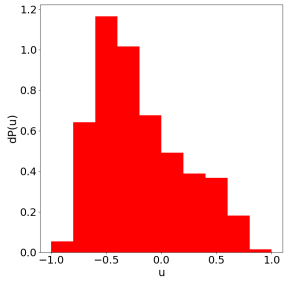


(c) Distribución de $u_{\vec{x}}^{t_0}$ ($s = 9202787456869569904$)

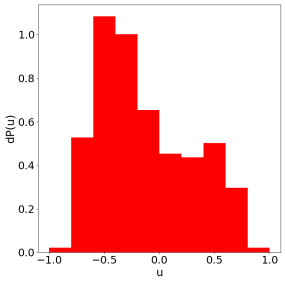
Figura E.3: Distribuciones de u para rejillas de mapeo acoplado de 64×64 sitios y acoplamiento $g = 0.204$ para diferentes semillas s. Las mediciones de `'u'` son efectuadas sobre el sistema completo al final de una evolución temporal de 2×10^5 iteraciones y un periodo transitorio de 10^3.



(a) Distribución de $u_{\vec{x}}^{t_0}$ ($s = 3340065878112466252$)



(b) Distribución de $u_{\vec{x}}^{t_0}$ ($s = 5805019659907812561$)



(c) Distribución de $u_{\vec{x}}^{t_0}$ ($s = 6378689186825912380$)

Figura E.4: Distribuciones de u para rejillas de mapeo acoplado de 64×64 sitios y acoplamiento $g = 0.204$ para diferentes semillas s. Las mediciones de `'u'` son efectuadas sobre el sistema completo al final de una evolución temporal de 2×10^5 iteraciones y un periodo transitorio de 10^4.

Apéndice F

Código para determinar comportamiento de los momentos de d

```
import random
import numpy as np
import networkx as nx
import matplotlib.pyplot as plt
import scipy
from scipy.stats import moment

##############################################################################
##############################################################################

### CÁLCULO DE DISTANCIAS ENTRE DOS ELEMENTOS DE UNA REJILLA CUADRADA
    BIDIMENSIONAL DOTADA DE CONDICIONTES DE FRONTERA PERIÓDICAS
def calc_dist(loc1, loc2, longitud):
  distx = loc1[0] - loc2[0]
  disty = loc1[1] - loc2[1]
  dist1 = (distx ** 2 + disty ** 2) ** (1 / 2)
  dist2 = ((distx - longitud) ** 2 + disty ** 2) ** (1 / 2)
  dist3 = ((distx + longitud) ** 2 + disty ** 2) ** (1 / 2)
  dist4 = (distx ** 2 + (disty - longitud) ** 2) ** (1 / 2)
  dist5 = (distx ** 2 + (disty + longitud) ** 2) ** (1 / 2)
  dist6 = ((distx + longitud) ** 2 + (disty + longitud) ** 2) ** (1 / 2)
  dist7 = ((distx - longitud) ** 2 + (disty + longitud) ** 2) ** (1 / 2)
  dist8 = ((distx - longitud) ** 2 + (disty - longitud) ** 2) ** (1 / 2)
  dist9 = ((distx + longitud) ** 2 + (disty - longitud) ** 2) ** (1 / 2)
  lista_distancias = [dist1, dist2, dist3, dist4, dist5, dist6, dist7, dist8
    , dist9]
  distancia = min(lista_distancias)
  return distancia

### INICIACIÓN DE GRÁFICA PARA SISTEMA CML
def ini_graf(longitud):
  grafica = nx.Graph()
  # Definición de nodos
  num_sitios = longitud ** 2
  for sitio in range(num_sitios):
    grafica.add_node(sitio, u = random.uniform(-1, 1))
  # Definición de bordes
  lista_edges = []
  for sitio1 in range(num_sitios):
    for sitio2 in range(num_sitios):
```

```
39        if sitio2 == sitio1:
40          continue
41        coord1 = (sitio1 % longitud, sitio1 // longitud)
42        coord2 = (sitio2 % longitud, sitio2 // longitud)
43        distancia = calc_dist(coord1, coord2, longitud)
44        lista_edges.append((sitio1, sitio2, {'path_distance': distancia}))
45    grafica.add_edges_from(lista_edges)
46    return grafica
47
48  ### CÁLCULO DE PRIMEROS VECINOS DE UN NODO EN UNA GRÁFICA CON BORDES DOTADOS
        DE ATRIBUTOS EQUIVALENTES A LAS DISTANCIAS ENTRE LOS SITIOS QUE LOS
      DEFINEN
49  def primeros_vecinos(grafica, nodo):
50    # Lista de posibles distancias
51    lista_dist = []
52    for vec1, datos1 in grafica.adj[nodo].items():
53      for keys1, dists1 in datos1.items():
54        lista_dist.append(dists1)
55    min_dist = min(lista_dist)
56    # Lista de sitios a distancia mínima (primeros vecinos)
57    lista_1v = []
58    for vec2, datos2 in grafica.adj[nodo].items():
59      for keys2, dists2 in datos2.items():
60        if dists2 == min_dist:
61          lista_1v.append(vec2)
62    return lista_1v
63
64  ### MAPEO PHI
65  def phi(valor_t):
66    if -1 <= valor_t < -1 / 3.:
67      valor_tt = (-3 * valor_t) - 2
68    elif -1 / 3. <= valor_t < 1 / 3.:
69      valor_tt = 3 * valor_t
70    elif 1 / 3. <= valor_t <= 1:
71      valor_tt = (-3 * valor_t) + 2
72    return valor_tt
73
74  ### EVOLUCIÓN TEMPORAL DE LA GRÁFICA (Y CÁLCULO DE PROMEDIO ESPACIAL DE U
      PARA CADA ITERACIÓN)
75  def ev_temp(num_iter, trans, g_acople, grafica, lista_1vecinos):
76    print('INICIO DE EVOLUCION TEMPORAL')
77    lista_sitios = list(grafica.nodes())
78    tenth_progress = int(num_iter / 10)
79    for iteracion1 in range(num_iter):
80      # Marcador de progreso
81      if iteracion1 % tenth_progress == 0:
82        print('*')
83      # Gráfica auxiliar con valores de 'u' de siguiente iteración
84      grafica_holder1 = nx.Graph()
85      for sitio1a in lista_sitios:
86        grafica_holder1.add_node(sitio1a, u = 0)
87        sum1vec1 = 0
88        for vecino1 in lista_1vecinos[sitio1a][1]:
89          dif_u1 = phi(grafica.nodes[vecino1]['u']) - phi(grafica.nodes[
      sitio1a]['u'])
90          sum1vec1 = sum1vec1 + dif_u1
91        grafica_holder1.nodes[sitio1a]['u'] = phi(grafica.nodes[sitio1a]['u'])
       + g_acople * sum1vec1
```

```
92      # Actualización de gráfica "original"
93      for sitio1b in lista_sitios:
94        grafica.nodes[sitio1b]['u'] = grafica_holder1.nodes[sitio1b]['u']
95      if iteracion1 <= trans - 1:
96        continue
97    print('FIN DE EVOLUCION TEMPORAL')
98    return grafica
99
100 ### IDENTIFICACIÓN DE LA SUBREJILLA A LA QUE PERTENECE UN SITIO PARTICULAR
      DE LA REJILLA
101 def ord_subrejilla(sitio, longitud, grano):
102   coord_sitio = (sitio % longitud, sitio // longitud)
103   coord_subrejilla = (coord_sitio[0] // grano, coord_sitio[1] // grano)
104   r = longitud /grano
105   num_subrejilla = int(coord_subrejilla[0] + coord_subrejilla[1]*r)
106   return num_subrejilla
107
108 ### CÁLCULO DE <|u|> PARA TODAS LAS SUBREJILLAS ASOCIADAS A UNA GRÁFICA
109 def calc_uG(grafica,grano):
110   num_sitios = len(list(grafica.nodes))
111   largo = num_sitios ** (1/2.)
112   total_subrejillas = int(num_sitios // (grano**2))
113   arr_absgrano = np.zeros(total_subrejillas)
114   arr_numgrano = np.zeros(total_subrejillas)
115   for sitio in range(num_sitios):
116     num_subrejilla = ord_subrejilla(sitio, largo, grano)
117     arr_absgrano[num_subrejilla] = arr_absgrano[num_subrejilla] + abs(
      grafica.nodes[sitio]['u'])
118     arr_numgrano[num_subrejilla] = arr_numgrano[num_subrejilla] + 1
119   arr_promabsgrano = np.zeros(total_subrejillas)
120   for subrejilla in range(total_subrejillas):
121     arr_promabsgrano[subrejilla] = arr_absgrano[subrejilla] / arr_numgrano[
      subrejilla]
122   return arr_promabsgrano
123
124 ##############################################################################
125 ##############################################################################
126
127 ### DEFINICIÓN DE PARÁMETROS DE SIMULACIÓN
128 s = int(input('Ingresa la semilla para números aleatorios (entero): '))
129 L = int(input('Ingresa la longitud de la rejilla CML (entero): '))
130 g = float(input('Ingresa la constante de acoplamiento (real positivo con
      tres decimales): '))
131 N_iter = int(input('Ingresa el total de iteraciones (entero): '))
132 transient = int(input('Ingresa el valor de transient (entero): '))
133 # Lista de granos
134 N_grains = int(input('Ingresa el total de longitudes de grano (entero): '))
135 G_values = np.zeros(N_grains)
136 for gr in range(N_grains):
137   G_values[gr] = int(input('Ingresa tamaño de grano (entero divisor de %(
      length)i): ' % {'length': L}))
138 print(G_values)
139 # Lista de exponentes
140 N_moments = int(input('Ingresa el total de momentos (entero): '))
141 moments_values = np.zeros(N_moments)
142 for mom in range(N_moments):
143   moments_values[mom] = int(input('Ingresa el momento (entero): '))
144 print(moments_values)
```

```
145
146 ### GENERACIÓN DE GRÁFICA Y LISTA CON PRIMEROS VECINOS
147 # Iniciación de generador de números aleatorios
148 random.seed(s)
149 # Definición de gráfica y lista con primeros vecinos
150 lattice = ini_graf(L)
151 list_1neighbors = []
152 for site in range(L**2):
153   list1v = primeros_vecinos(lattice, site)
154   list_1neighbors.append((site, list1v))
155
156 ### EVOLUCIÓN TEMPORAL DE SISTEMA CML
157 lattice = ev_temp(N_iter, transient, g, lattice, list_1neighbors)
158
159 ### CÁLCULO DE MOMENTOS PARA DIFERENTES LONGITUDES DE SUBREJILLAS
160 # Arreglo para momentos calculados con SciPy
161 arr_moments_sci = np.zeros((N_grains, N_moments))
162 for counter_grain, G_size in enumerate(G_values):
163   arr_avgabsG = calc_uG(lattice,G_size)
164   for counter_moment, m_value in enumerate(moments_values):
165     # Cálculo con SciPy
166     arr_moments_sci[counter_grain][counter_moment] = moment(arr_avgabsG,
    moment=m_value)
167   print('Grano concluido: ' +  str(float(counter_grain)))
168 # Almacenamiento
169 fname_sci = 'tesis_Egolf_DM_L%(length)i_g%(coupling).3f_Niter%(iterations).3
    e_trans%(trans).3e_seed%(seed)i_MomentsSciPy.txt'
170 np.savetxt(fname_sci % dict_fname, arr_moments_sci)
171
172 ### GRÁFICA CON MOMENTOS
173 plt.figure(1)
174 fig1, ax1B = plt.subplots(nrows = 1, ncols = 1, figsize = (10,10))
175 plt.tight_layout(pad=8, h_pad=4, w_pad=8)
176 #Transpuestas de arreglos con momentos
177 arr_moments_sci_graph = np.transpose(arr_moments_sci)
178 # Comportamiento esperado para momentos pares
179 max_G = max(G_values)
180 min_G = min(G_values)
181 fst_G = (max_G - min_G) / 2.
182 G_ref = np.arange(fst_G,max_G,0.1)
183 s_ref = np.zeros(len(G_ref))
184 for gs in range(len(G_ref)):
185   s_ref[gs] = 1 / (G_ref[gs] ** 2)
186 # Gráfica con momentos calculados con SciPy
187 ax1B.set_yscale('log')
188 ax1B.set_ylabel(r'$\sigma^{2}$', fontsize=22)
189 ax1B.set_xscale('log')
190 ax1B.set_xlabel('Tamaño de grano ' + r'$\it{G}$', fontsize=22)
191 ax1B.tick_params(axis='both', which='major', labelsize=20)
192 ax1B.plot(G_values, 2*arr_moments_sci_graph[0], 'b^--', label=r'$2 \langle d
    ^{2} \rangle $', markersize=18, linewidth=8)
193 ax1B.plot(G_values, (abs(arr_moments_sci_graph[1]))**(2/3.), 'bo:', label=r'
    $\langle d^{3} \rangle^{2/3}$', markersize=18, linewidth=8)
194 ax1B.plot(G_values, (4*arr_moments_sci_graph[2]/3.)**(1/2.), 'rs:', label=r'
    $ \left( 4 \langle d^{4} \rangle / 3 \right)^{1/2} $', markersize=18,
    linewidth=8)
195 ax1B.plot(G_values, (8*arr_moments_sci_graph[3]/15.)**(1/3.), 'g^--', label=
    r'$\left( 8 \langle d^{6} \rangle / 15 \right)^{1/3}$', markersize=18,
```

```
    linewidth=8)
ax1B.plot(G_ref, s_ref, 'k-', label=r'$G^{-2}$', linewidth=6)
ax1B.legend(loc='lower left', fontsize=20)
# Almacenamiento
imgname = 'tesis_Egolf_DM_L%(length)i_g%(coupling).3f_Niter%(iterations).3
    e_trans%(trans).3e_seed%(seed)i_Graph_hbfs22.png'
dict_imgname = {'length': L, 'coupling': g, 'iterations': N_iter, 'trans':
    transient, 'seed': s}
plt.savefig(imgname % dict_imgname)

print('Programa concluido')
```

Apéndice G

Gráficas de simulaciones para prueba de comportamiento Gaussiano de momentos de desviaciones d

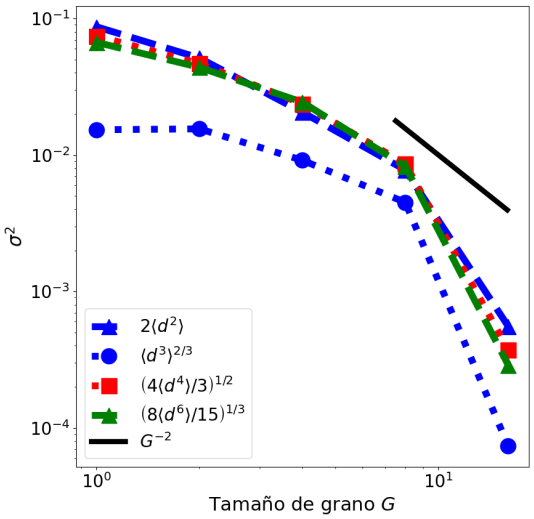


Figura G.1: Momentos de las desviaciones $d = (\langle|u|\rangle_G - \langle|u|\rangle_{x,t})$ (con factores de escala usados por Egolf) para una rejilla de mapeo acoplado de 32×32 sitios, con un acoplamiento $g = 0.195$ y granos de tamaño $G = [1, 2, 4, 8, 16]$. El promedio $\langle|u|\rangle_{x,t}$ fue calculado sobre un intervalo temporal de 2×10^5 iteraciones.

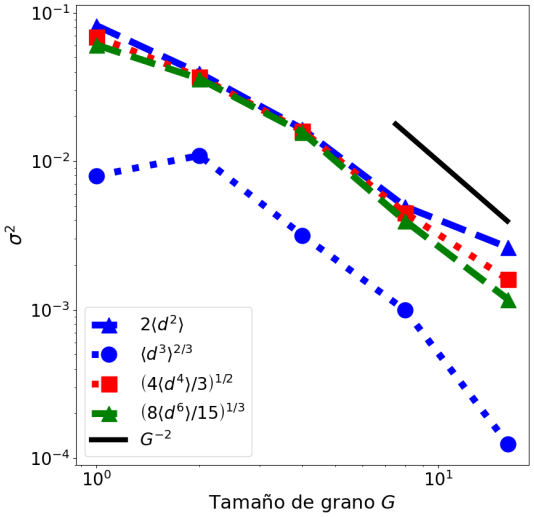


Figura G.2: Momentos de las desviaciones $d = (\langle|u|\rangle_G - \langle|u|\rangle_{x,t})$ (con factores de escala usados por Egolf) para una rejilla de mapeo acoplado de 32×32 sitios, con un acoplamiento $g = 0.195$ y granos de tamaño $G = [1, 2, 4, 8, 16]$. El promedio $\langle|u|\rangle_{x,t}$ fue calculado sobre un intervalo temporal de 5×10^5 iteraciones.

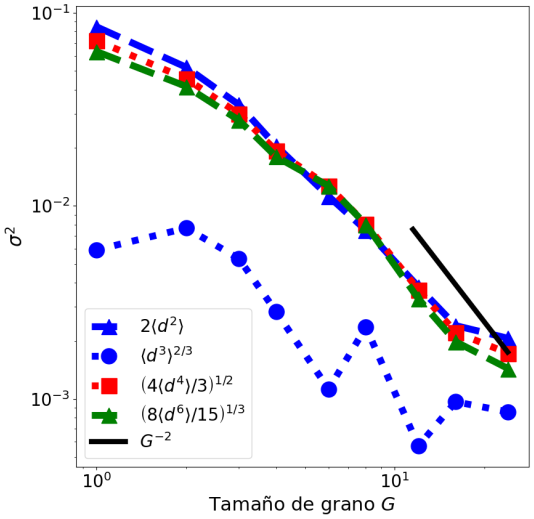


Figura G.3: Momentos de las desviaciones $d = (\langle|u|\rangle_G - \langle|u|\rangle_{x,t})$ (con factores de escala usados por Egolf) para una rejilla de mapeo acoplado de 48×48 sitios, con un acoplamiento $g = 0.195$ y granos de tamaño $G = [1, 2, 3, 4, 6, 8, 12, 16, 24]$. El promedio $\langle|u|\rangle_{x,t}$ fue calculado sobre un intervalo temporal de 2×10^5 iteraciones.

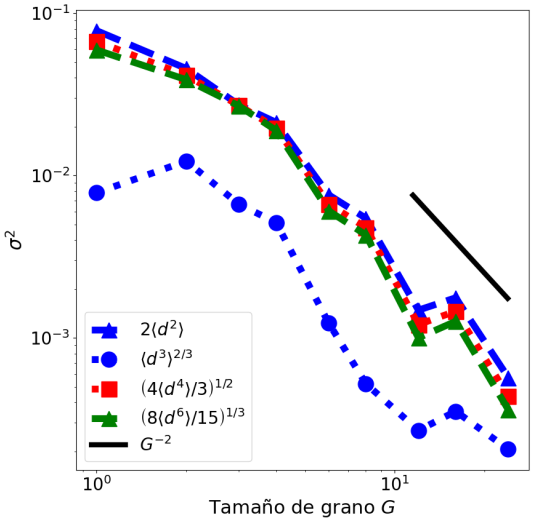


Figura G.4: Momentos de las desviaciones $d = (\langle|u|\rangle_G - \langle|u|\rangle_{x,t})$ (con factores de escala usados por Egolf) para una rejilla de mapeo acoplado de 48×48 sitios, con un acoplamiento $g = 0.195$ y granos de tamaño $G = [1, 2, 3, 4, 6, 8, 12, 16, 24]$. El promedio $\langle|u|\rangle_{x,t}$ fue calculado sobre un intervalo temporal de 5×10^5 iteraciones.

Apéndice H

Código para prueba de balance detallado en rejilla de mapeo acoplado a grano grueso

```
import sys
import matplotlib.pyplot as plt
import numpy as np
import networkx as nx
import random as rand
import json
import csv

##############################################################################
##############################################################################

### CÁLCULO DE DISTANCIAS ENTRE ELEMENTOS DE REJILLA CML BAJO CONDICIONES DE FRONTERA PERIÓDICA
def calc_dist(loc1,loc2,longitud):
  dist1 = ((loc1[0] - loc2[0])**2 + (loc1[1] - loc2[1])**2)**(1/2)
  dist2 = ((loc1[0] - longitud - loc2[0])**2 + (loc1[1] - loc2[1])**2)**(1/2)
  dist3 = ((loc1[0] + longitud - loc2[0])**2 + (loc1[1] - loc2[1])**2)**(1/2)
  dist4 = ((loc1[0] - loc2[0])**2 + (loc1[1] - longitud - loc2[1])**2)**(1/2)
  dist5 = ((loc1[0] - loc2[0])**2 + (loc1[1] + longitud - loc2[1])**2)**(1/2)
  dist6 = ((loc1[0] + longitud - loc2[0])**2 + (loc1[1] + longitud - loc2[1])**2)**(1/2)
  dist7 = ((loc1[0] - longitud - loc2[0])**2 + (loc1[1] + longitud - loc2[1])**2)**(1/2)
  dist8 = ((loc1[0] - longitud - loc2[0])**2 + (loc1[1] - longitud - loc2[1])**2)**(1/2)
  dist9 = ((loc1[0] + longitud - loc2[0])**2 + (loc1[1] - longitud - loc2[1])**2)**(1/2)
  lista_distancias = [dist1, dist2, dist3, dist4, dist5, dist6, dist7, dist8, dist9]
  distancia = min(lista_distancias)
  return distancia

### DEFINICIÓN DE GRÁFICA PARA REJILLA CML
```

```
28 def ini_graf(longitud):
29   grafica = nx.Graph()
30   num_sitios = longitud**2
31   # Definición de nodos
32   for sitio in range(num_sitios):
33     grafica.add_node(sitio, u = rand.uniform(-1,1))
34   # Definición de edges
35   lista_edges = []
36   for sitio1 in range(num_sitios):
37     for sitio2 in range(num_sitios):
38       if sitio2 == sitio1:
39         continue
40       coord1 = (sitio1 % longitud, sitio1 // longitud)
41       coord2 = (sitio2 % longitud, sitio2 // longitud)
42       distancia = calc_dist(coord1,coord2,longitud)
43       lista_edges.append((sitio1,sitio2,{'path_distance': distancia}))
44   grafica.add_edges_from(lista_edges)
45   return grafica
46
47 ### CÁLCULO DE PRIMEROS VECINOS PARA UN NODO PARTICULAR EN LA GRÁFICA
48 def primeros_vecinos(graf,nodo):
49   lista_dist = []
50   for vec1, datos1 in graf.adj[nodo].items():
51     for keys1, dists1 in datos1.items():
52       lista_dist.append(dists1)
53   dist_min = min(lista_dist)
54   lista_1v = []
55   for vec2, datos2 in graf.adj[nodo].items():
56     for keys2, dists2 in datos2.items():
57       if dists2 == dist_min:
58         lista_1v.append(vec2)
59   return dist_min, lista_1v
60
61 ### MAPEO PHI
62 def phi(valor_t):
63   if -1 <= valor_t < -1/3.:
64     valor_tt = (-3*valor_t) - 2
65   elif -1/3. <= valor_t < 1/3.:
66     valor_tt = 3*valor_t
67   elif 1/3. <= valor_t <= 1:
68     valor_tt = (-3*valor_t) + 2
69   return valor_tt
70
71 ### EVOLUCIÓN TEMPORAL DE LA REJILLA
72 def ev_temp(num_iter,trans,x0,g_acople,grafica,lista_1vecinos,guardar):
73   lista_sitios = list(grafica.nodes())
74   if guardar == True:
75     arr_promtemp = np.zeros((num_iter-trans))
76     for iteracion1 in range(num_iter):
77       grafica_holder1 = nx.Graph()
78       for sitio1 in lista_sitios:
79         grafica_holder1.add_node(sitio1, u = 0)
80         sum1vec1 = 0
81         for vecino1 in lista_1vecinos[sitio1][1]:
82           dif_u1 = phi(grafica.nodes[vecino1]['u']) - phi(grafica.nodes[
    sitio1]['u'])
83           sum1vec1 = sum1vec1 + dif_u1
84         grafica_holder1.nodes[sitio1]['u'] = phi(grafica.nodes[sitio1]['u'])
```

```
    + g_acople*sum1vec1
      for sitio1b in lista_sitios:
        grafica.nodes[sitio1b]['u'] = grafica_holder1.nodes[sitio1b]['u']
      if iteracion1 <= trans - 1:
        continue
      arr_promtemp[iteracion1 - trans] = grafica.nodes[x0]['u']
    return grafica, arr_promtemp
  else:
    for iteracion2 in range(num_iter):
      grafica_holder2 = nx.Graph()
      for sitio2 in lista_sitios:
        grafica_holder2.add_node(sitio2, u = 0)
        sum1vec2 = 0
        for vecino2 in lista_1vecinos[sitio2][1]:
          dif_u2 = phi(grafica.nodes[vecino2]['u']) - phi(grafica.nodes[
   sitio2]['u'])
          sum1vec2 = sum1vec2 + dif_u2
        grafica_holder2.nodes[sitio2]['u'] = phi(grafica.nodes[sitio2]['u'])
    + g_acople*sum1vec2
      for sitio2b in lista_sitios:
        grafica.nodes[sitio2b]['u'] = grafica_holder2.nodes[sitio2b]['u']
    return grafica

### IDENTIFICACIÓN DE LA SUBREJILLA DE GXG SITIOS A LA QUE PERTENECE UN
   SITIO PARTICULAR
def ord_subrejilla(sitio,longitud,grano):
  coord_sitio = (sitio % longitud, sitio // longitud)
  coord_subrejilla = (coord_sitio[0]//grano, coord_sitio[1]//grano)
  r = longitud/grano
  num_subrejilla = int(coord_subrejilla[0] + coord_subrejilla[1]*r)
  return num_subrejilla

### FUNCIÓN SIGNO MODIFICADA
def sign_modif(arreglo_prom_espacial,intervalo,contador):
  prom_st = np.sign(sum(arreglo_prom_espacial) / intervalo)
  if prom_st == -1:
    return 0, contador
  elif prom_st == 1:
    return 1, contador
  elif prom_st == 0:
    if contador == 0:
      contador = 1
      return 0, contador
    elif contador == 1:
      contador = 0
      return 1, contador

##############################################################################
##############################################################################

### BALANCE DETALLADO I - DEFINICIÓN DE REJILLA CML Y "TERMALIZACIÓN"
# Selección aleatoria de semilla
s = rand.randrange(sys.maxsize)
# Definición de otros parámetros
L = int(input('Ingresa la longitud de la rejilla (entero): '))
Gs = int(input('Ingresa el tamaño del grano (entero): '))
N_iter = int(input('Ingresa el total de iteraciones (entero): '))
transient = int(input('Ingresa el valor del transient (entero): '))
```

```
g = float(input('Ingresa la constante de acoplamiento (real con tres
    decimales): '))
DeltaT = int(input('Ingresa el periodo temporal DeltaT (entero): '))
N_turns = int(input('Ingresa el número de rondas (entero): '))
# Iniciación de generador de números aleatorios
rand.seed(s)
# Definición de gráfica y lista con primeros vecinos
G = ini_graf(L)
list_min_dist = []
list_1neighbors = []
for site in range(int(L**2)):
  dist, list1v = primeros_vecinos(G,site)
  list_min_dist.append(dist)
  list_1neighbors.append((site,list1v))
# Evolución temporal de rejilla
safe_db = False
G = ev_temp(N_iter, transient, 0, g, G, list_1neighbors, safe_db)
print('Evolución temporal concluida')

# Balance Detallado II - Evolución temporal de rejilla a grano grueso
total_subG = (L**2) // (Gs**2)
arr_avgsubG = np.zeros(total_subG)
arr_finsubG = np.zeros(total_subG)
counter_zero = 0
arr_tavg_subG = np.zeros((total_subG,DeltaT))
#list_t_states = np.zeros(L**2)
list_CG_states = np.zeros(N_turns)

#fname_listofstates_t = "tesis_Egolf_baldet_L%(length)i_G%(grainsize)i_g%(
    coupling).3f_Niter%(iterations).3e_trans%(trans).3e_Nrondas%(turns).3
    e_seed%(seed)i_ListOfStates_t.csv"
fname_listofstates_CG = "tesis_Egolf_baldet_L%(length)i_G%(grainsize)i_g%(
    coupling).3f_Niter%(iterations).3e_trans%(trans).3e_Nrondas%(turns).3
    e_seed%(seed)i_ListOfStates_CG.txt"
dict_fname = {'length': L, 'grainsize': Gs, 'coupling': g, 'iterations':
    N_iter, 'trans': transient, 'turns': N_turns, 'seed': s}

tenth_progress = int(N_turns / 10)

for loop in range(N_turns):
# print('Inicio de Ronda: ' + str(float(loop)))
  ind_progress = loop % tenth_progress
  if ind_progress == 0:
    print('.')
  for t in range(DeltaT):
    arr_subG = np.zeros(total_subG)
    arr_nsub = np.zeros(total_subG)
    # Cálculo de promedio espacial sobre subrejillas
    for nod in range(L**2):
      n_subG = ord_subrejilla(nod,L,Gs)
      arr_subG[n_subG] = arr_subG[n_subG] + G.nodes[nod]['u']
      arr_nsub[n_subG] = arr_nsub[n_subG] + 1
    for subgrid in range(total_subG):
      arr_avgsubG[subgrid] = arr_subG[subgrid] / arr_nsub[subgrid]
    arr_tavg_subG[:,t] = arr_avgsubG
    # Update de rejilla CML
    graph_holder = nx.Graph()
    for element in range(L**2):
```

```
191       graph_holder.add_node(element, u = 0)
192       sum_neighbors = 0
193       for nextnode in list_1neighbors[element][1]:
194         diff = phi(G.nodes[nextnode]['u']) - phi(G.nodes[element]['u'])
195         sum_neighbors = sum_neighbors + diff
196       graph_holder.nodes[element]['u'] = phi(G.nodes[element]['u']) + g *
      sum_neighbors
197     for elementb in range(L**2):
198       G.nodes[elementb]['u'] = graph_holder.nodes[elementb]['u']
199     # Definición de lista de estados de rejilla "fina"
200 #   for elementc in range(L**2):
201 #     list_t_states[elementc] = graph_holder.nodes[elementc]['u']
202 #   with open(fname_listofstates_t % dict_fname, 'a') as t_states_file:
203 #     t_states_writer = csv.writer(t_states_file, delimiter=',', quotechar
      ='"', quoting=csv.QUOTE_MINIMAL)
204 #     t_states_writer.writerow(list_t_states)
205   # Cálculo de promedio temporal sobre intervalo DeltaT
206   for sg in range(total_subG):
207     arr_finsubG[sg], counter_zero = sign_modif(arr_tavg_subG[sg],DeltaT,
      counter_zero)
208   # Definición de lista de estados de rejilla a grano grueso
209   sum_binary_id = 0
210   for entry in range(total_subG):
211     if arr_finsubG[entry] == 0:
212       continue
213     elif arr_finsubG[entry] == 1:
214       term = 2**(entry)
215       sum_binary_id = sum_binary_id + term
216 # print('Ronda ' + str(float(loop)) + ': ' + str(arr_finsubG))
217   list_CG_states[loop] = sum_binary_id
218
219 np.savetxt(fname_listofstates_CG % dict_fname, list_CG_states)
220 read_arr_CGstates = np.loadtxt(fname_listofstates_CG % dict_fname)
221
222 print("Evolución temporal de rejilla a grano grueso concluida")
223
224 # Balance Detallado III - Matriz de transiciones
225 matrix_transitions = np.zeros((2**(total_subG),2**(total_subG)))
226 for state in range(1,N_turns):
227   matrix_row = int(list_CG_states[state-1])
228   matrix_column = int(list_CG_states[state])
229   matrix_transitions[matrix_row][matrix_column] = matrix_transitions[
      matrix_row][matrix_column] + 1
230
231 print("Matriz de transiciones definida")
232
233 # Balance Detallado IV - Diccionario para graficar promedios <(Delta/N)^2>
234 fname_dictorig = "tesis_Egolf_baldet_L%(length)i_G%(grainsize)i_g%(coupling)
      .3f_Niter%(iterations).3e_trans%(trans).3e_Nrondas%(turns).3e_seed%(seed)
      i_Dict_Original.txt"
235 dict_db = {}
236 for row in range(2**(total_subG) - 1):
237   for column in range(row + 1, 2**(total_subG)):
238     pos_trans = matrix_transitions[row][column]
239     neg_trans = matrix_transitions[column][row]
240     delta = pos_trans - neg_trans
241     total = pos_trans + neg_trans
242     if total == 0:
```

```
      continue
    key_total = 'a' + str(total)
    if key_total in dict_db:
      dict_db[key_total][0] = dict_db[key_total][0] + delta ** 2
      dict_db[key_total][1] = dict_db[key_total][1] + 1
    else:
      dict_db[key_total] = [delta ** 2, 1]

with open(fname_dictorig % dict_fname, "w") as write_dicto:
  json.dump(dict_db, write_dicto)

print("Diccionario 1 definido")

# Balance Detallado VI - Cálculo y gráfica de promedios <(Delta/N)^2> (Guardar Imagen)
M = len(read_dicto)
arr_dbo_N = np.zeros(M)
arr_dbo_avg = np.zeros(M)
counter_entry_dbo = 0
for key in dict_db:
  valueN = int(float(key[1:]))
  arr_dbo_N[counter_entry_dbo] = valueN
  n_entries = dict_db[key][1]
  # Promedio "a mano"
  sum_avg = dict_db[key][0] / (valueN ** 2)
  avg = sum_avg / n_entries
  arr_dbo_avg[counter_entry_dbo] = avg
  counter_entry_dbo = counter_entry_dbo + 1

# Cálculo de chi^2
chi2 = 0
for counter_N, vN in enumerate(arr_dbo_N):
  theoretical = 1.0 / vN
  diff = (arr_dbo_avg[counter_N] - theoretical) ** 2
  chi2 = chi2 + diff

max_N = max(arr_dbo_N)
x_ref = np.arange(1,max_N,0.1)
y_ref = np.zeros(len(x_ref))
for x in range(len(x_ref)):
  y_ref[x] = 1 / x_ref[x]

figC2, axC2 = plt.subplots(nrows=1, ncols=1, figsize=(15,15))
plt.rcParams.update({'font.size': 24})
plt.rcParams.update({'xtick.labelsize': 22})
plt.rcParams.update({'ytick.labelsize': 22})
plt.tight_layout(pad=4, h_pad=4, w_pad=4)
axC2.set_title('Balance Detallado - %(m)i valores de ' % {'m': M} + r'$N$' + ', ' + r'$ \chi^{2} = %(dif).3e $' % {'dif': chi2}, size=18)
axC2.set_yscale('log')
axC2.set_ylabel(r'$\left\langle \left( \Delta / N \right)^{2} \right\rangle$', fontsize=22)
axC2.set_xscale('log')
axC2.set_xlabel('N', fontsize=22)
axC2.plot(x_ref, y_ref, 'r-', label=r'$N^{-1}$', markersize=18, linewidth=10)
axC2.plot(arr_dbo_N, arr_dbo_avg, 'bo', label='P_np', markersize=18)
axC2.legend(loc='lower left', fontsize='small')
```

```
297
298 plt.savefig("tesis_Egolf_baldet_L%(length)i_G%(grainsize)i_g%(coupling).3
       f_Niter%(iterations).3e_trans%(trans).3e_Nrondas%(turns).3e_seed%(seed)
       i_Graph.png" % dict_fname)
299 print('Programa concluido')
```

Apéndice I

Código para calcular promedios sobre *bins* con una anchura de crecimiento exponencial

```
import matplotlib.pyplot as plt
import numpy as np
import networkx as nx
import random as rand
import json
import csv

##############################################################################
##############################################################################

### SOLICITUD DE PARÁMETROS
# Caracterísiticas de rejilla
L = int(input('Ingresa la longitud de la rejilla (entero): '))
Gs = int(input('Ingresa el tamaño del grano (entero): '))
g = float(input('Ingresa la constante de acoplamiento (real con tres
    decimales): '))
N_iter = int(input('Ingresa el total de iteraciones (entero): '))
transient = int(input('Ingresa el valor del transient (entero): '))
N_turns = int(input('Ingresa el número de rondas (entero): '))
seed = int(input('Ingresa el valor de la semilla: '))
# Tasa de exponencial
a = float(input('Ingresa la tasa de crecimiento de bin: '))
comp_keys = {'length': L, 'grainsize': Gs, 'coupling': g, 'iterations':
    N_iter, 'trans': transient, 'turns': N_turns, 's': seed, 'rate': a }

### RECUPERACIÓN DE DICCIONARIO
comp_fname = "./Res_BD_L%(length)i_G%(grainsize)i_g%(coupling).3f_Niter%(
    iterations).3e_trans%(trans).3e_Nrondas%(turns).3e_seed%(s)i/
    tesis_Egolf_baldet_L%(length)i_G%(grainsize)i_g%(coupling).3f_Niter%(
    iterations).3e_trans%(trans).3e_Nrondas%(turns).3e_seed%(s)
    i_Dict_Original.txt"
with open(comp_fname % comp_keys, "r") as read_comp:
  dict_comp = json.load(read_comp)

### "BINNING" EXPONENCIAL
M = len(dict_comp)
# Límites en número de transiciones
```

```
list_string_keys = list(dict_comp.keys())
list_keys = []
for string_key in list_string_keys:
  list_keys.append(int(float(string_key[1:])))
min_N = min(list_keys)
max_N = max(list_keys)
print('Máximo número de transiciones: ' + str(float(max_N)))
list_edges = [min_N]
# Edges
bin_step = 0
while max(list_edges) < max_N:
  factor = np.exp(a * bin_step)
  sup_lim = max(list_edges) + factor
  list_edges.append(sup_lim)
  bin_step = bin_step + 1
  print(list_edges)
# Promedios sobre bins
total_bins = len(list_edges) - 1
arr_previs = np.zeros([total_bins, 2])
for key_N in dict_comp:
  # Cálculo de suma y conteo de factores (Delta/N)^2
  value_N = int(float(key_N[1:]))
  total_dN2 = dict_comp[key_N][1]
  sum_dN2 = dict_comp[key_N][0] / (value_N ** 2)
  # Identificación de bin
  for counter_edge, edge in enumerate(list_edges[1:], start=1):
    if edge >= value_N:
      arr_previs[counter_edge - 1] = arr_previs[counter_edge - 1] + np.array
   ([sum_dN2, total_dN2])
      break
# Almacenamiento
previs_fname = "tesis_Egolf_baldet_L%(length)i_G%(grainsize)i_g%(coupling).3
   f_Niter%(iterations).3e_trans%(trans).3e_Nrondas%(turns).3e_seed%(s)
   i_tasa%(rate).3f_Previs.txt"
np.savetxt( previs_fname % comp_keys, arr_previs)

### GRÁFICA
# Arreglo con centros de bins
center_bins = np.zeros(total_bins)
for index in range(total_bins):
  bot_lim = list_edges[index]
  top_lim = list_edges[index + 1]
  dif = top_lim - bot_lim
  center_bins[index] = bot_lim + dif / 2.
# Arreglo con promedios sobre bins
avg_bins = np.zeros(total_bins)
for index in range(total_bins):
  if arr_previs[index][1] == 0:
    continue
  else:
    avg_bins[index] = arr_previs[index][0] / arr_previs[index][1]
print(M)
print(avg_bins)
# Cálculo de chi^2
chi2 = 0
for counter_N, vN in enumerate(center_bins):
  if vN == 0:
    continue
```

```
  else:
    theoretical = 1.0 / vN
      diff = (avg_bins[counter_N] - theoretical) ** 2
      chi2 = chi2 + diff

# Comportamiento esperado
x_ref = np.arange(1,max_N,0.1)
y_ref = np.zeros(len(x_ref))
for x in range(len(x_ref)):
  y_ref[x] = 1 / x_ref[x]

figC2, axC2 = plt.subplots(nrows=1, ncols=1, figsize=(10,10))
plt.rcParams.update({'font.size': 24})
plt.rcParams.update({'xtick.labelsize': 32})
plt.rcParams.update({'ytick.labelsize': 42})
plt.tight_layout(pad=4, h_pad=4, w_pad=4)

axC2.set_title('Balance Detallado - %(m)i bins ' % {'m': total_bins} + ', '
    + r'$ \chi^{2} = %(dif).3e $' % {'dif': chi2}, size=18)
axC2.set_yscale('log')
axC2.set_ylabel(r'$\left\langle \left( \Delta / N \right)^{2} \right\rangle$
    ', fontsize=22)
axC2.set_xscale('log')
axC2.set_xlabel(r'$N$', fontsize=22)
axC2.plot(x_ref, y_ref, 'r-', label=r'$N^{-1}$', markersize=18, linewidth=8)
axC2.plot(center_bins, avg_bins, 'b^', label='Exp',  markerfacecolor='none',
     markeredgewidth=2, markersize=18)
axC2.legend(loc='lower left', fontsize='small')

plt.savefig("tesis_Egolf_baldet_L%(length)i_G%(grainsize)i_g%(coupling).3
    f_Niter%(iterations).3e_trans%(trans).3e_Nrondas%(turns).3e_seed%(s)
    i_tasa%(rate).3f_Graph.png" % comp_keys)
print('Programa concluido')
```

Apéndice J

Código para prueba de distribución de Boltzmann para los estados de rejillas de mapeo acoplado “a grano grueso”

```
import sys
import numpy as np
import networkx as nx
import random
import matplotlib.pyplot as plt
import scipy
from scipy.optimize import minimize
from scipy.optimize import Bounds
from scipy.optimize import LinearConstraint
import json

##############################################################################
##############################################################################

# Cálculo de distancias entre elementos en una rejilla cuadrada
    bidimensional bajo condiciones de frontera periódicas
def calc_dist(loc1, loc2, longitud):
  distx = loc1[0] - loc2[0]
  disty = loc1[1] - loc2[1]
  dist1 = (distx ** 2 + disty ** 2) ** (1 / 2.0)
  dist2 = ((distx - longitud) ** 2 + disty ** 2) ** (1 / 2.0)
  dist3 = ((distx + longitud) ** 2 + disty ** 2) ** (1 / 2.0)
  dist4 = (distx ** 2 + (disty - longitud) ** 2) ** (1 / 2.0)
  dist5 = (distx ** 2 + (disty + longitud) ** 2) ** (1 / 2.0)
  dist6 = ((distx + longitud) ** 2 + (disty + longitud) ** 2) ** (1 / 2.0)
  dist7 = ((distx - longitud) ** 2 + (disty + longitud) ** 2) ** (1 / 2.0)
  dist8 = ((distx - longitud) ** 2 + (disty - longitud) ** 2) ** (1 / 2.0)
  dist9 = ((distx + longitud) ** 2 + (disty - longitud) ** 2) ** (1 / 2.0)
  lista_distancias = [dist1, dist2, dist3, dist4, dist5, dist6, dist7, dist8
    , dist9]
  distancia = min(lista_distancias)
  return distancia

# Iniciación de gráfica para rejilla CML a grano grueso en estado 0
def ini_graf(longitud):
  grafica = nx.Graph()
  num_sitios = longitud**2
```

```
  # Definición de nodos
  for sitio in range(num_sitios):
    grafica.add_node(sitio, u = -1)
  # Definición de edges
  lista_edges = []
  for sitio1 in range(num_sitios):
    for sitio2 in range(num_sitios):
      if sitio2 == sitio1:
        continue
      coord1 = (sitio1 % longitud, sitio1 // longitud)
      coord2 = (sitio2 % longitud, sitio2 // longitud)
      distancia = calc_dist(coord1, coord2, longitud)
      lista_edges.append((sitio1, sitio2, {'path_distance': distancia}))
  grafica.add_edges_from(lista_edges)
  return grafica

# Configuración de gráfica en un estado definido
def def_edo(estado, grafica):
  referencia = estado
  exp = 0
  while referencia > 0:
    factor2 = 2 ** (exp)
    tasa = referencia / factor2
    if tasa >= 1:
      exp = exp + 1
    else:
      grafica.nodes[exp - 1]['u'] = 1
      factorneg = 2 ** (exp - 1)
      referencia = referencia - factorneg
      exp = 0
  return grafica

# Definición de lista con distancias posibles entre sitios de la gráfica
def list_dists(grafica):
  lista_distancias = []
  for (nodo1, nodo2, distancia) in grafica.edges.data('path_distance'):
    if distancia not in lista_distancias:
      lista_distancias.append(distancia)
    else:
      continue
  lista_distancias = sorted(lista_distancias)
  return lista_distancias

### PROBABILIDADES DE BOLTZMANN PARA ENERGÍAS
def prob_boltzmann(arreglo_alfas):
  # Definición de diccionario con coeficientes alfas
  diccionario_alfas = {}
  for dist_iter in range(len(site_dists)):
    clave_dist = list_key_dists[dist_iter]
    diccionario_alfas[clave_dist] = arreglo_alfas[dist_iter]
  # Cálculo de probabilidades de Boltzmann para todos los estados diferentes
    que están disponibles al sistema
  p_calc = np.zeros(len(list_sums))
  funcion_Z = 0
  contador_p = 0
  for entry in protoE:
    # Cálculo de Hamiltoniano para un estado particular dadas ciertas alfas
    hamiltoniano = 0
```

```
93      for distancia in list_key_dists:
94        factor = diccionario_alfas[distancia] * entry[0][distancia]
95        hamiltoniano = hamiltoniano + factor
96      # Definición (parcial) de probabilidad para cada estado
97      p_calc[contador_p] = entry[2] * np.exp(-hamiltoniano)
98      # Definición (gradual) de función de partición Z
99      funcion_Z = funcion_Z + (entry[2] * np.exp(-hamiltoniano))
100     contador_p = contador_p + 1
101   # Definición de probabilidad de Boltzmann para todos los estados
      disponibles al sistema
102   p_calc = p_calc / funcion_Z
103   return p_calc, funcion_Z
104
105 # Mínimos cuadrados
106 def min_cuadrados(arreglo_alfas):
107   prob_calc, funcion_Z = prob_boltzmann(arreglo_alfas)
108   dif = 0
109   for i in range(len(prob_calc)):
110     dif = dif + ((prob_calc[i] - protoE[i][1])) ** 2
111   return dif
112
113 ### PROBABILIDADES DE BOLTZMANN PARA ESTADOS
114 def pe_estado(arreglo_alfas):
115   # Definición de diccionario con coeficientes alfas
116   diccionario_alfas = {}
117   for dist_iter in range(len(site_dists)):
118     clave_dist = list_key_dists[dist_iter]
119     diccionario_alfas[clave_dist] = arreglo_alfas[dist_iter]
120   # Cálculo de probabilidades de Boltzmann para todos los estados diferentes
      que están disponibles al sistema
121   p_edo = np.zeros(N_total_states)
122   e_edo = np.zeros(N_total_states)
123   funcion_Z = 0
124   contador_p = 0
125   for estado in range(N_total_states):
126     clave = 's' + str(float(estado))
127     # Cálculo de Hamiltoniano para un estado particular dadas ciertas alfas
128     hamiltoniano = 0
129     for distancia in list_key_dists:
130       factor = diccionario_alfas[distancia] * dict_sums[clave][distancia]
131       hamiltoniano = hamiltoniano + factor
132     # Definición (parcial) de probabilidad para cada estado
133     p_edo[contador_p] = np.exp(-hamiltoniano)
134     # Definición de energía para cada estado
135     e_edo[contador_p] = hamiltoniano
136     # Definición (gradual) de función de partición Z
137     funcion_Z = funcion_Z + np.exp(-hamiltoniano)
138     contador_p = contador_p + 1
139   # Definición de probabilidad de Boltzmann para todos los estados
      disponibles al sistema
140   p_edo = p_edo / funcion_Z
141   return e_edo, p_edo, funcion_Z
142
143 ### FUNCIÓN PARA ORDENAR ARREGLOS
144 def ordenador(energias_ajuste, probabilidades_ajuste,
      probabilidades_experimentales):
145   # Ordenamiento de energías resultantes de ajuste
146   energias_ordenadas = sorted(energias_ajuste)
```

```
147   # Identificación de índices
148   indices = []
149   for energia_ord in energias_ordenadas:
150     for ind, energia in enumerate(energias_ajuste):
151       if energia == energia_ord:
152         indices.append(ind)
153         break
154   # Ordenamiento de probabilidades
155   prob_ajuste_ordenadas = []
156   prob_exp_ordenadas = []
157   for ind in indices:
158     prob_ajuste_ordenadas.append(probabilidades_ajuste[ind])
159     prob_exp_ordenadas.append(probabilidades_experimentales[ind])
160   return energias_ordenadas, prob_ajuste_ordenadas, prob_exp_ordenadas
161
162 ##############################################################################
163 ##############################################################################
164
165 ### RECUPERACIÓN DE EVOLUCIÓN DE REJILLA CML DE GRANO GRUESO
166 # Solicitud de parámetros de rejilla CML
167 L = int(input('Ingresa la longitud de la rejilla (entero): '))
168 G = int(input('Ingresa el tamaño de grano (entero): '))
169 N_total_sites = int(L ** 2 / G ** 2)
170 N_total_states = 2 ** (N_total_sites)
171 g = float(input('Ingresa la constante de acoplamiento (real con tres
      decimales): '))
172 N_iter = int(input('Ingresa el total de iteraciones (entero): '))
173 transient = int(input('Ingresa el valor de transient (entero): '))
174 N_turns = int(input('Ingresa el total de rondas (entero): '))
175 n_tries = int(input('Ingresa el número de ansatz (entero): '))
176 seed = int(input('Ingresa el valor de la semilla (entero): '))
177 c = float(input('Ingresa desplazamiento (real): '))
178 sref = random.randrange(sys.maxsize)
179 random.seed(sref)
180 # Lectura de archivo con datos
181 fname_listofstates_CG = './Res_BD_L%(length)i_G%(grainsize)i_g%(coupling).3
      f_Niter%(iterations).3e_trans%(trans).3e_Nrondas%(turns).3e_seed%(s)i/
      tesis_Egolf_baldet_L%(length)i_G%(grainsize)i_g%(coupling).3f_Niter%(
      iterations).3e_trans%(trans).3e_Nrondas%(turns).3e_seed%(s)
      i_ListOfStates_CG.txt'
182 dict_read = {'length': L, 'grainsize': G, 'coupling': g, 'iterations':
      N_iter, 'trans': transient, 'turns': N_turns, 's': seed}
183 # Definición de arreglo con evolución de rejilla CML de grano grueso
184 arr_CG_states = np.loadtxt(fname_listofstates_CG % dict_read)
185
186 ### DISTRIBUCIÓN DE PROBABILIDAD DADA LA EVOLUCIÓN DE LA REJILLA CML A GRANO
      GRUESO
187 print('Histograma - INICIO')
188 hist_CG, bins_CG = np.histogram(arr_CG_states, bins = N_total_states, range
      = (0, N_total_states), density = True)
189 # Almacenamiento de histograma (probabilidades "experimentales")
190 filename_hist = './tesis_Egolf_EGb_Basic_L%(length)i_G%(grainsize)i_g%(
      coupling).3f_Niter%(iterations).3e_trans%(trans).3e_Nrondas%(turns).3
      e_Nansatz%(ansatz)i_seed%(s)i_c%(ct).3f_sr%(ref)i_LeastSquares09_HistDB.
      txt'
191 dict_fname = {'length': L, 'grainsize': G, 'coupling': g, 'iterations':
      N_iter, 'trans': transient, 'turns': N_turns, 'ansatz': n_tries, 's':
      seed, 'ct': c, 'ref': sref}
```

```
192 np.savetxt(filename_hist % dict_fname, hist_CG)
193 print('Histograma - FINAL')
194
195 ### DICCIONARIO CON ESTADOS S_I Y SUMAS DE PRODUCTOS ENTRE PARES DE ESPINES
    (INDEXADAS POR DISTANCIAS ENTRE SITIOS)
196 LCG = int(N_total_sites ** (1 / 2.0))
197 lattice = ini_graf(LCG)
198 print('Diccionario con proto-energías - INICIO')
199 dict_sums = {}
200 tenth_progress_sums = int(N_total_states / 10)
201 for state in range(N_total_states):
202   # Marcador de progreso
203   if state % tenth_progress_sums == 0:
204     print('*')
205   # Definición de gráfica en estado definido
206   lattice = def_edo(state,lattice)
207   # Cálculo de productos
208   dict_prod = {}
209   for site1 in range(N_total_sites - 1):
210     for site2 in range(site1 + 1, N_total_sites):
211       product = lattice.nodes[site1]['u'] * lattice.nodes[site2]['u']
212       distance = lattice.edges[site1, site2]['path_distance']
213       key = 'd' + str(float(distance))
214       if key in dict_prod:
215         dict_prod[key].append(product)
216       else:
217         dict_prod[key] = [product]
218   # Cálculo de sumas de productos
219   dict_sums_holder = {}
220   for key in dict_prod:
221     dict_sums_holder[key] = sum(dict_prod[key])
222   # Definición de diccionario final
223   key_state = 's' + str(float(state))
224   dict_sums[key_state] = dict_sums_holder
225   lattice = ini_graf(LCG)
226 # Almacenamiento de diccionario
227 filename_dictsums = './tesis_Egolf_EGb_Basic_L%(length)i_G%(grainsize)i_g%(
    coupling).3f_Niter%(iterations).3e_trans%(trans).3e_Nrondas%(turns).3
    e_Nansatz%(ansatz)i_seed%(s)i_c%(ct).3f_sr%(ref)
    i_LeastSquares09_ProtoE_dict.txt'
228 with open(filename_dictsums % dict_fname, "w") as write_dictsums:
229   json.dump(dict_sums, write_dictsums)
230 print('Diccionario con proto-energías - FINAL')
231
232 ### LISTA CON SUMAS DE PRODUCTOS ENTRE PARES DE ESPÍNES (PROTO-ENERGÍAS)
233 print('Lista con proto-energías (sin repeticiones) - INICIO')
234 list_sums = []
235 for sum_prod in dict_sums.values():
236   if sum_prod not in list_sums:
237     list_sums.append(sum_prod)
238   else:
239     continue
240 # Almacenamiento de lista con sumas
241 filename_listsums = './tesis_Egolf_EGb_Basic_L%(length)i_G%(grainsize)i_g%(
    coupling).3f_Niter%(iterations).3e_trans%(trans).3e_Nrondas%(turns).3
    e_Nansatz%(ansatz)i_seed%(s)i_c%(ct).3f_sr%(ref)
    i_LeastSquares09_ProtoE_list.txt'
242 with open(filename_listsums % dict_fname, "w") as write_listsums:
```

```
  json.dump(list_sums, write_listsums)
print('Lista con proto-energías (sin repeticiones) - FINAL')

### LISTA CON PROTO-ENERGÍAS, PROBABILIDADES Y NÚMERO DE ESTADOS CORRESPONDIENTES
print('Lista con proto-energías y probabilidades - INICIO')
protoE_desord = []
for sum_prod in list_sums:
  index_holder = []
  for key_sum in dict_sums:
    if dict_sums[key_sum] == sum_prod:
      state = int(float(key_sum[1:]))
      index_holder.append(state)
    else:
      continue
  num_states_H = len(index_holder)
  sum_probs = 0
  for index in index_holder:
    term = hist_CG[index]
    sum_probs = sum_probs + term
  protoE_desord.append([sum_prod, sum_probs, num_states_H])
# Almacenamiento de lista con probabilidades y total de estados correspondientes a proto-energías
filename_protoE_desord = './tesis_Egolf_EGb_Basic_L%(length)i_G%(grainsize)i_g%(coupling).3f_Niter%(iterations).3e_trans%(trans).3e_Nrondas%(turns).3e_Nansatz%(ansatz)i_seed%(s)i_c%(ct).3f_sr%(ref)i_LeastSquares09_ProtoE&Prob&NS_desordenadas.txt'
with open(filename_protoE_desord % dict_fname, "w") as write_protoE_desord:
  json.dump(protoE_desord, write_protoE_desord)
print('Lista con probabilidades de Hamiltoniano - FINAL')

### ORDENAMIENTO DE PROTO-ENERGÍAS POR PROBABILIDAD
# Lista con probabilidades de proto-energías, ordenadas de mayor a menor
prob_ord = []
for pair in protoE_desord:
  prob_ord.append(pair[1])
prob_ord = sorted(prob_ord, reverse = True)
# Ordenamiento de lista con proto-energías y probabilidades asociadas
protoE = []
print('Ordenamiento de proto-energías por probabilidades - INICIO')
for probability in prob_ord:
  for entry in protoE_desord:
    if entry[1] == probability:
      if entry not in protoE:
        protoE.append(entry)
        break
      else:
        continue
    else:
      continue
# Almacenamiento de lista con probabilidades y total de estados correspondientes a proto-energías
filename_protoE = './tesis_Egolf_EGb_Basic_L%(length)i_G%(grainsize)i_g%(coupling).3f_Niter%(iterations).3e_trans%(trans).3e_Nrondas%(turns).3e_Nansatz%(ansatz)i_seed%(s)i_c%(ct).3f_sr%(ref)i_LeastSquares09_ProtoE&Prob&NS.txt'
with open(filename_protoE % dict_fname, "w") as write_protoE:
  json.dump(protoE, write_protoE)
```

```
print('Ordenamiento de proto-energías por probabilidades - FINAL')

### RECUPERACIÓN DE LISTA CON DISTANCIAS ENTRE SITIOS
# Lista con distancias
site_dists = list_dists(lattice)
# Lista con claves de distancias
list_key_dists = []
for distance in site_dists:
  list_key_dists.append('d' + str(distance))

### DEFINICIÓN DE RESTRICCIONES SOBRE PROTO-ENERGÍAS
# Definición de arreglo con restricciones
print('Definición de restricciones sobre proto-energías - INICIO')
num_restrict = len(protoE)
num_dists = len(list_key_dists)
ineq_protoE = np.zeros((num_restrict, num_dists))
for n_restriction in range(num_restrict):
  for n_distance in range(num_dists):
    key_dist = list_key_dists[n_distance]
    ineq_protoE[n_restriction][n_distance] = protoE[n_restriction][0][
   key_dist]
# Definición de fronteras
lower_limit = np.zeros(num_restrict)
upper_limit = np.zeros(num_restrict)
for indx in range(len(upper_limit)):
  upper_limit[indx] = np.inf
# Definición de restricción (lineal)
linear_constraint = LinearConstraint(ineq_protoE, lower_limit, upper_limit)
print('Definición de restricciones sobre proto-energías - FINAL')

##############################################################################
### MÍNIMOS CUADRADOS (CON ESQUEMAS LOCAL Y GLOBAL)
##############################################################################

######################### ESQUEMA LOCAL 0 (DEFAULT)############################

### MÍNIMOS CUADRADOS - DEFAULT
print('Mínimos cuadrados (default) - INICIO')
ansatz_df = np.zeros((n_tries, len(site_dists)))
alphas_df = np.zeros((n_tries, len(site_dists)))
chi2_df = np.zeros(n_tries)
# Minimización con múltiples ansatz
for a_df in range(n_tries):
  # Definición de ansatz
  for value_df in range(len(site_dists)):
    ansatz_df[a_df][value_df] = random.uniform(-1,1)
  # Minimización
  solution_def = minimize(min_cuadrados, ansatz_df[a_df], options={'disp':
   True})
  alphas_df[a_df] = solution_def.x
  chi2_df[a_df] = min_cuadrados(alphas_df[a_df])
# Almacenamiento
ansatz_df_fname = 'tesis_Egolf_EGb_Chi2Std_ProbE_L%(length)i_G%(grainsize)
   i_g%(coupling).3f_Niter%(iterations).3e_trans%(trans).3e_Nrondas%(turns)
   .3e_Nansatz%(ansatz)i_seed%(s)i_c%(ct).3f_sr%(ref)
   i_LeastSquares09_Default_Ansatz.txt'
np.savetxt(ansatz_df_fname % dict_fname, ansatz_df)
alphas_df_fname = 'tesis_Egolf_EGb_Chi2Std_ProbE_L%(length)i_G%(grainsize)
```

```
    i_g%(coupling).3f_Niter%(iterations).3e_trans%(trans).3e_Nrondas%(turns)
    .3e_Nansatz%(ansatz)i_seed%(s)i_c%(ct).3f_sr%(ref)
    i_LeastSquares09_Default_TotalAlphas.txt'
np.savetxt(alphas_df_fname % dict_fname, alphas_df)
chi2_df_fname = 'tesis_Egolf_EGb_Chi2Std_ProbE_L%(length)i_G%(grainsize)i_g
    %(coupling).3f_Niter%(iterations).3e_trans%(trans).3e_Nrondas%(turns).3
    e_Nansatz%(ansatz)i_seed%(s)i_c%(ct).3f_sr%(ref)
    i_LeastSquares09_Default_Chi2.txt'
np.savetxt(chi2_df_fname % dict_fname, chi2_df)
# Identificación de chi2 mínima
min_chi2_df = np.nanmin(chi2_df)
for counter_df, v_chi2_df in enumerate(chi2_df):
  if v_chi2_df == min_chi2_df:
    min_ansatz_df = counter_df
# Alphas correspondiente a mínimo
alphas_min_def = alphas_df[min_ansatz_df]
# Cálculo de probabilidades
pBoltz_def, Z_def = prob_boltzmann(alphas_min_def)
Z_def = np.array([Z_def])
px_def = np.zeros(len(protoE))
for counter, prob in enumerate(pBoltz_def):
  px_def[counter] = prob*c
# Almacenamiento
fname_def = 'tesis_Egolf_EGb_Chi2Std_ProbE_L%(length)i_G%(grainsize)i_g%(
    coupling).3f_Niter%(iterations).3e_trans%(trans).3e_Nrondas%(turns).3
    e_Nansatz%(ansatz)i_seed%(s)i_c%(ct).3f_sr%(ref)
    i_LeastSquares09_Default_ProbBoltz.txt'
np.savetxt(fname_def % dict_fname, pBoltz_def)
fname_Z_def = 'tesis_Egolf_EGb_Chi2Std_ProbE_L%(length)i_G%(grainsize)i_g%(
    coupling).3f_Niter%(iterations).3e_trans%(trans).3e_Nrondas%(turns).3
    e_Nansatz%(ansatz)i_seed%(s)i_c%(ct).3f_sr%(ref)
    i_LeastSquares09_Default_Z.txt'
np.savetxt(fname_Z_def % dict_fname, Z_def)
print('Mínimos cuadrados (default) - FINAL')

print('Diferencia con valores finales: ' + str(min_cuadrados(alphas_min_def)
    ))

### DICCIONARIO CON COEFICIENTES ALFA FINALES - DEFAULT
print('Diccionario con coeficientes alfa finales (default) - INICIO')
dict_alphas_def = {}
for n_distance in range(len(site_dists)):
  k_distance = list_key_dists[n_distance]
  dict_alphas_def[k_distance] = alphas_min_def[n_distance]
# Almacenamiento
fname_alphas_def = 'tesis_Egolf_EGb_Chi2Std_ProbE_L%(length)i_G%(grainsize)
    i_g%(coupling).3f_Niter%(iterations).3e_trans%(trans).3e_Nrondas%(turns)
    .3e_Nansatz%(ansatz)i_seed%(s)i_c%(ct).3f_sr%(ref)
    i_LeastSquares09_Default_Alphas.txt'
with open(fname_alphas_def % dict_fname, "w") as write_dict_alphas_def:
  json.dump(dict_alphas_def, write_dict_alphas_def)
print('Diccionario con coeficientes alfa finales (default) - FINAL')

### ENERGÍAS FINALES - DEFAULT
print('Lista con Hammiltonianos finales (default) - INICIO')
final_H_def = np.zeros(len(protoE))
prob_exp = np.zeros(len(protoE))
counter_H_def = 0
```

```
for entry in protoE:
  H = 0
  for dist in entry[0]:
    term = dict_alphas_def[dist] * entry[0][dist]
    H = H + term
  final_H_def[counter_H_def] = H
  prob_exp[counter_H_def] = entry[1]
  counter_H_def = counter_H_def + 1
# Almacenamiento
fname_H_def = 'tesis_Egolf_EGb_Chi2Std_ProbE_L%(length)i_G%(grainsize)i_g%(coupling).3f_Niter%(iterations).3e_trans%(trans).3e_Nrondas%(turns).3e_Nansatz%(ansatz)i_seed%(s)i_c%(ct).3f_sr%(ref)i_LeastSquares09_Default_H.txt'
np.savetxt(fname_H_def % dict_fname, final_H_def)
print('Lista con Hammiltonianos finales (default) - FINAL')
# Ordenamiento
final_H_def_gf, pBoltz_def_gf, prob_exp_def_gf = ordenador(final_H_def, pBoltz_def, prob_exp)
final_H_def_gf, px_def_gf, prob_exp_def_gf = ordenador(final_H_def, px_def, prob_exp)

### ESQUEMA ESTADO
es_def, ps_def, Zs_def = pe_estado(alphas_min_def)
Zs_def = np.array([Zs_def])
# Almacenamiento
state_p_def = 'tesis_Egolf_EGb_Chi2Std_ProbE_L%(length)i_G%(grainsize)i_g%(coupling).3f_Niter%(iterations).3e_trans%(trans).3e_Nrondas%(turns).3e_Nansatz%(ansatz)i_seed%(s)i_c%(ct).3f_sr%(ref)i_LeastSquares09_Default_ProbBoltzState.txt'
np.savetxt(state_p_def % dict_fname, ps_def)
state_H_def = 'tesis_Egolf_EGb_Chi2Std_ProbE_L%(length)i_G%(grainsize)i_g%(coupling).3f_Niter%(iterations).3e_trans%(trans).3e_Nrondas%(turns).3e_Nansatz%(ansatz)i_seed%(s)i_c%(ct).3f_sr%(ref)i_LeastSquares09_Default_HState.txt'
np.savetxt(state_H_def % dict_fname, es_def)
state_Z_def = 'tesis_Egolf_EGb_Chi2Std_ProbE_L%(length)i_G%(grainsize)i_g%(coupling).3f_Niter%(iterations).3e_trans%(trans).3e_Nrondas%(turns).3e_Nansatz%(ansatz)i_seed%(s)i_c%(ct).3f_sr%(ref)i_LeastSquares09_Default_ZState.txt'
np.savetxt(state_Z_def % dict_fname, Zs_def)
# Ordenamiento
es_def_gf, ps_def_gf, hCG_def_gf = ordenador(es_def, ps_def, hist_CG)


######################## ESQUEMA LOCAL 1 ('trust-constr')####################

### MÍNIMOS CUADRADOS - LOCAL 1
print('Mínimos cuadrados (local - \'trust-constr\') - INICIO')
ansatz_l1 = np.zeros((n_tries, len(site_dists)))
alphas_l1 = np.zeros((n_tries, len(site_dists)))
chi2_l1 = np.zeros(n_tries)
# Minimización con múltiples ansatz
for a_l1 in range(n_tries):
  # Definición de ansatz
  for value_l1 in range(len(site_dists)):
    ansatz_l1[a_l1][value_l1] = random.uniform(-1,1)
  # Minimización
  solution_l1 = minimize(min_cuadrados, ansatz_l1[a_l1], options={'disp':
```

```
    True})
  alphas_l1[a_l1] = solution_l1.x
  chi2_l1[a_l1] = min_cuadrados(alphas_l1[a_l1])
# Almacenamiento
ansatz_l1_fname = 'tesis_Egolf_EGb_Chi2Std_ProbE_L%(length)i_G%(grainsize)
    i_g%(coupling).3f_Niter%(iterations).3e_trans%(trans).3e_Nrondas%(turns)
    .3e_Nansatz%(ansatz)i_seed%(s)i_c%(ct).3f_sr%(ref)
    i_LeastSquares09_Local1_Ansatz.txt'
np.savetxt(ansatz_l1_fname % dict_fname, ansatz_l1)
alphas_l1_fname = 'tesis_Egolf_EGb_Chi2Std_ProbE_L%(length)i_G%(grainsize)
    i_g%(coupling).3f_Niter%(iterations).3e_trans%(trans).3e_Nrondas%(turns)
    .3e_Nansatz%(ansatz)i_seed%(s)i_c%(ct).3f_sr%(ref)
    i_LeastSquares09_Local1_TotalAlphas.txt'
np.savetxt(alphas_l1_fname % dict_fname, alphas_l1)
chi2_l1_fname = 'tesis_Egolf_EGb_Chi2Std_ProbE_L%(length)i_G%(grainsize)i_g
    %(coupling).3f_Niter%(iterations).3e_trans%(trans).3e_Nrondas%(turns).3
    e_Nansatz%(ansatz)i_seed%(s)i_c%(ct).3f_sr%(ref)
    i_LeastSquares09_Local1_Chi2.txt'
np.savetxt(chi2_l1_fname % dict_fname, chi2_l1)
# Identificación de chi2 mínima
min_chi2_l1 = np.nanmin(chi2_l1)
for counter_l1, v_chi2_l1 in enumerate(chi2_l1):
  if v_chi2_l1 == min_chi2_l1:
    min_ansatz_l1 = counter_l1
# Alphas correspondiente a mínimo
alphas_min_l1 = alphas_l1[min_ansatz_l1]
# Cálculo de probabilidades
pBoltz_l1, Z_l1 = prob_boltzmann(alphas_min_l1)
Z_l1 = np.array([Z_l1])
px_l1 = np.zeros(len(protoE))
for counter, prob in enumerate(pBoltz_l1):
  px_l1[counter] = prob*c
# Almacenamiento
fname_l1 = 'tesis_Egolf_EGb_Chi2Std_ProbE_L%(length)i_G%(grainsize)i_g%(
    coupling).3f_Niter%(iterations).3e_trans%(trans).3e_Nrondas%(turns).3
    e_Nansatz%(ansatz)i_seed%(s)i_c%(ct).3f_sr%(ref)
    i_LeastSquares09_Local1_ProbBoltz.txt'
np.savetxt(fname_l1 % dict_fname, pBoltz_l1)
fname_Z_l1 = 'tesis_Egolf_EGb_Chi2Std_ProbE_L%(length)i_G%(grainsize)i_g%(
    coupling).3f_Niter%(iterations).3e_trans%(trans).3e_Nrondas%(turns).3
    e_Nansatz%(ansatz)i_seed%(s)i_c%(ct).3f_sr%(ref)i_LeastSquares09_Local1_Z
    .txt'
np.savetxt(fname_Z_l1 % dict_fname, Z_l1)
print('Diferencia con valores finales: ' + str(min_cuadrados(alphas_min_l1))
    )
print('Mínimos cuadrados (local - \'trust-constr\') - FINAL')

### DICCIONARIO CON COEFICIENTES ALFA FINALES - LOCAL 1
print('Diccionario con coeficientes alfa finales (local - \'trust-constr\')
    - INICIO')
dict_alphas_l1 = {}
for n_distance in range(len(site_dists)):
  k_distance = list_key_dists[n_distance]
  dict_alphas_l1[k_distance] = alphas_min_l1[n_distance]
# Almacenamiento
fname_alphas_l1 = 'tesis_Egolf_EGb_Chi2Std_ProbE_L%(length)i_G%(grainsize)
    i_g%(coupling).3f_Niter%(iterations).3e_trans%(trans).3e_Nrondas%(turns)
    .3e_Nansatz%(ansatz)i_seed%(s)i_c%(ct).3f_sr%(ref)
```

```
    i_LeastSquares09_Local1_Alphas.txt'
with open(fname_alphas_l1 % dict_fname, "w") as write_dict_alphas_l1:
  json.dump(dict_alphas_l1, write_dict_alphas_l1)
print('Diccionario con coeficientes alfa finales (local - \'trust-constr\')
    - FINAL')

### ENERGÍAS FINALES - LOCAL 1
print('Lista con Hamiltonianos finales (local - \'trust-constr\') - INICIO')
final_H_l1 = np.zeros(len(protoE))
counter_H_l1 = 0
for entry in protoE:
  H = 0
  for dist in entry[0]:
    term = dict_alphas_l1[dist] * entry[0][dist]
    H = H + term
  final_H_l1[counter_H_l1] = H
  counter_H_l1 = counter_H_l1 + 1
# Almacenamiento
fname_H_l1 = 'tesis_Egolf_EGb_Chi2Std_ProbE_L%(length)i_G%(grainsize)i_g%(
    coupling).3f_Niter%(iterations).3e_trans%(trans).3e_Nrondas%(turns).3
    e_Nansatz%(ansatz)i_seed%(s)i_c%(ct).3f_sr%(ref)i_LeastSquares09_Local1_H
    .txt'
np.savetxt(fname_H_l1 % dict_fname, final_H_l1)
print('Lista con Hamiltonianos finales (local - \'trust-constr\') - FINAL')
# Ordenamiento
final_H_l1_gf, pBoltz_l1_gf, prob_exp_l1_gf = ordenador(final_H_l1,
    pBoltz_l1, prob_exp)
final_H_l1_gf, px_l1_gf, prob_exp_l1_gf = ordenador(final_H_l1, px_l1,
    prob_exp)

### ESQUEMA ESTADO
es_l1, ps_l1, Zs_l1 = pe_estado(alphas_min_l1)
Zs_l1 = np.array([Zs_l1])
# Almacenamiento
state_p_l1 = 'tesis_Egolf_EGb_Chi2Std_ProbE_L%(length)i_G%(grainsize)i_g%(
    coupling).3f_Niter%(iterations).3e_trans%(trans).3e_Nrondas%(turns).3
    e_Nansatz%(ansatz)i_seed%(s)i_c%(ct).3f_sr%(ref)
    i_LeastSquares09_Local1_ProbBoltzState.txt'
np.savetxt(state_p_l1 % dict_fname, ps_l1)
state_H_l1 = 'tesis_Egolf_EGb_Chi2Std_ProbE_L%(length)i_G%(grainsize)i_g%(
    coupling).3f_Niter%(iterations).3e_trans%(trans).3e_Nrondas%(turns).3
    e_Nansatz%(ansatz)i_seed%(s)i_c%(ct).3f_sr%(ref)
    i_LeastSquares09_Local1_HState.txt'
np.savetxt(state_H_l1 % dict_fname, es_l1)
state_Z_l1 = 'tesis_Egolf_EGb_Chi2Std_ProbE_L%(length)i_G%(grainsize)i_g%(
    coupling).3f_Niter%(iterations).3e_trans%(trans).3e_Nrondas%(turns).3
    e_Nansatz%(ansatz)i_seed%(s)i_c%(ct).3f_sr%(ref)
    i_LeastSquares09_Local1_ZState.txt'
np.savetxt(state_Z_l1 % dict_fname, Zs_l1)
# Ordenamiento
es_l1_gf, ps_l1_gf, hCG_l1_gf = ordenador(es_l1, ps_l1, hist_CG)


########################## ESQUEMA GRADUADO 0 (DEFAULT)######################

### MÍNIMOS CUADRADOS - GRADUADO DEFAULT
print('Mínimos cuadrados (graduado - default) - INICIO')
total_alphas = len(site_dists)
```

```
ansatz_gdf = np.zeros((n_tries, len(site_dists)))
alphas_gdf = np.zeros((n_tries, len(site_dists)))
chi2_gdf = np.zeros(n_tries)
# Minimización con múltiples ansatz
for a_gdf in range(n_tries):
  # Definición de ansatz
  for value_gdf in range(len(site_dists)):
    ansatz_gdf[a_gdf][value_gdf] = random.uniform(-1,1)
  # Minimización gradual
  for turn in range(total_alphas):
    # Coeficientes fijos
    lower_alphas = np.zeros(total_alphas)
    upper_alphas = np.zeros(total_alphas)
    for n_alpha in range(total_alphas):
      if n_alpha == turn:
        lower_alphas[n_alpha] = -np.inf
        upper_alphas[n_alpha] = np.inf
      else:
        lower_alphas[n_alpha] = alphas_gdf[a_gdf][n_alpha]
        upper_alphas[n_alpha] = alphas_gdf[a_gdf][n_alpha]
    bounds_alphas = Bounds(lower_alphas, upper_alphas)
    # Minimización
    solution_grad_def = minimize(min_cuadrados, ansatz_gdf[a_gdf], options={
    'disp': True})
    alphas_gdf[a_gdf] = solution_grad_def.x
    chi2_gdf[a_gdf] = min_cuadrados(alphas_gdf[a_gdf])
# Almacenamiento
ansatz_gdf_fname = 'tesis_Egolf_EGb_Chi2Std_ProbE_L%(length)i_G%(grainsize)
    i_g%(coupling).3f_Niter%(iterations).3e_trans%(trans).3e_Nrondas%(turns)
    .3e_Nansatz%(ansatz)i_seed%(s)i_c%(ct).3f_sr%(ref)
    i_LeastSquares09_G_Default_Ansatz.txt'
np.savetxt(ansatz_gdf_fname % dict_fname, ansatz_gdf)
alphas_gdf_fname = 'tesis_Egolf_EGb_Chi2Std_ProbE_L%(length)i_G%(grainsize)
    i_g%(coupling).3f_Niter%(iterations).3e_trans%(trans).3e_Nrondas%(turns)
    .3e_Nansatz%(ansatz)i_seed%(s)i_c%(ct).3f_sr%(ref)
    i_LeastSquares09_G_Default_TotalAlphas.txt'
np.savetxt(alphas_gdf_fname % dict_fname, alphas_gdf)
chi2_gdf_fname = 'tesis_Egolf_EGb_Chi2Std_ProbE_L%(length)i_G%(grainsize)i_g
    %(coupling).3f_Niter%(iterations).3e_trans%(trans).3e_Nrondas%(turns).3
    e_Nansatz%(ansatz)i_seed%(s)i_c%(ct).3f_sr%(ref)
    i_LeastSquares09_G_Default_Chi2.txt'
np.savetxt(chi2_gdf_fname % dict_fname, chi2_gdf)
# Identificación de chi2 mínima
min_chi2_gdf = np.nanmin(chi2_gdf)
for counter_gdf, v_chi2_gdf in enumerate(chi2_gdf):
  if v_chi2_gdf == min_chi2_gdf:
    min_ansatz_gdf = counter_gdf
# Alphas correspondiente a mínimo
alphas_grad_def = alphas_gdf[min_ansatz_gdf]
# Cálculo de probabilidades
pBoltz_grad_def, Z_grad_def = prob_boltzmann(alphas_grad_def)
Z_grad_def = np.array([Z_grad_def])
px_grad_def = np.zeros(len(protoE))
for counter, prob in enumerate(pBoltz_grad_def):
  px_grad_def[counter] = prob*c
# Almacenamiento
fname_grad_def = 'tesis_Egolf_EGb_Chi2Std_ProbE_L%(length)i_G%(grainsize)i_g
    %(coupling).3f_Niter%(iterations).3e_trans%(trans).3e_Nrondas%(turns).3
```

```
    e_Nansatz%(ansatz)i_seed%(s)i_c%(ct).3f_sr%(ref)
    i_LeastSquares09_G_Default_ProbBoltz.txt'
np.savetxt(fname_grad_def % dict_fname, pBoltz_grad_def)
fname_Z_grad_def = 'tesis_Egolf_EGb_Chi2Std_ProbE_L%(length)i_G%(grainsize)
    i_g%(coupling).3f_Niter%(iterations).3e_trans%(trans).3e_Nrondas%(turns)
    .3e_Nansatz%(ansatz)i_seed%(s)i_c%(ct).3f_sr%(ref)
    i_LeastSquares09_G_Default_Z.txt'
np.savetxt(fname_Z_grad_def % dict_fname, Z_grad_def)
print('Diferencia con valores finales: ' + str(min_cuadrados(alphas_grad_def
    )))
print('Mínimos cuadrados (graduado - default) - FINAL')

### DICCIONARIO CON COEFICIENTES ALFA FINALES - GRADUADO DEFAULT
print('Diccionario con coeficientes alfa finales (graduado - default) -
    INICIO')
dict_alphas_grad_def = {}
for n_distance in range(total_alphas):
  k_distance = list_key_dists[n_distance]
  dict_alphas_grad_def[k_distance] = alphas_grad_def[n_distance]
# Almacenamiento
fname_alphas_grad_def = 'tesis_Egolf_EGb_Chi2Std_ProbE_L%(length)i_G%(
    grainsize)i_g%(coupling).3f_Niter%(iterations).3e_trans%(trans).3
    e_Nrondas%(turns).3e_Nansatz%(ansatz)i_seed%(s)i_c%(ct).3f_sr%(ref)
    i_LeastSquares09_G_Default_Alphas.txt'
with open(fname_alphas_grad_def % dict_fname, "w") as
    write_dict_alphas_grad_def:
  json.dump(dict_alphas_grad_def, write_dict_alphas_grad_def)
print('Diccionario con coeficientes alfa finales (graduado - default) -
    FINAL')

### ENERGÍAS FINALES - GRADUADO DEFAULT
print('Lista con Hammiltonianos finales (graduado - default) - INICIO')
final_H_grad_def = np.zeros(len(protoE))
counter_H_grad_def = 0
for entry in protoE:
  H = 0
  for dist in entry[0]:
    term = dict_alphas_grad_def[dist] * entry[0][dist]
    H = H + term
  final_H_grad_def[counter_H_grad_def] = H
  counter_H_grad_def = counter_H_grad_def + 1
# Almacenamiento
fname_H_grad_def = 'tesis_Egolf_EGb_Chi2Std_ProbE_L%(length)i_G%(grainsize)
    i_g%(coupling).3f_Niter%(iterations).3e_trans%(trans).3e_Nrondas%(turns)
    .3e_Nansatz%(ansatz)i_seed%(s)i_c%(ct).3f_sr%(ref)
    i_LeastSquares09_G_Default_H.txt'
np.savetxt(fname_H_grad_def % dict_fname, final_H_grad_def)
print('Lista con Hammiltonianos finales (graduado - default) - FINAL')
# Ordenamiento
final_H_grad_def_gf, pBoltz_grad_def_gf, prob_exp_grad_def_gf = ordenador(
    final_H_grad_def, pBoltz_grad_def, prob_exp)
final_H_grad_def_gf, px_grad_def_gf, prob_exp_grad_def_gf = ordenador(
    final_H_grad_def, px_grad_def, prob_exp)

### ESQUEMA ESTADO
es_graddef, ps_graddef, Zs_graddef = pe_estado(alphas_grad_def)
Zs_graddef = np.array([Zs_graddef])
# Almacenamiento
```

```
599 state_p_graddef = 'tesis_Egolf_EGb_Chi2Std_ProbE_L%(length)i_G%(grainsize)
    i_g%(coupling).3f_Niter%(iterations).3e_trans%(trans).3e_Nrondas%(turns)
    .3e_Nansatz%(ansatz)i_seed%(s)i_c%(ct).3f_sr%(ref)
    i_LeastSquares09_G_Default_ProbBoltzState.txt'
600 np.savetxt(state_p_graddef % dict_fname, ps_graddef)
601 state_H_graddef = 'tesis_Egolf_EGb_Chi2Std_ProbE_L%(length)i_G%(grainsize)
    i_g%(coupling).3f_Niter%(iterations).3e_trans%(trans).3e_Nrondas%(turns)
    .3e_Nansatz%(ansatz)i_seed%(s)i_c%(ct).3f_sr%(ref)
    i_LeastSquares09_G_Default_HState.txt'
602 np.savetxt(state_H_graddef % dict_fname, es_graddef)
603 state_Z_graddef = 'tesis_Egolf_EGb_Chi2Std_ProbE_L%(length)i_G%(grainsize)
    i_g%(coupling).3f_Niter%(iterations).3e_trans%(trans).3e_Nrondas%(turns)
    .3e_Nansatz%(ansatz)i_seed%(s)i_c%(ct).3f_sr%(ref)
    i_LeastSquares09_G_Default_ZState.txt'
604 np.savetxt(state_Z_graddef % dict_fname, Zs_graddef)
605 # Ordenamiento
606 es_graddef_gf, ps_graddef_gf, hCG_graddef_gf = ordenador(es_graddef,
    ps_graddef, hist_CG)
607
608
609 ################### ESQUEMA GRADUADO 1 ('trust-constr') ###################
610
611 ### MÍNIMOS CUADRADOS - GRADUADO L1
612 print('Mínimos cuadrados (graduado - \'trust-constr\') - INICIO')
613 total_alphas = len(site_dists)
614 ansatz_gl1 = np.zeros((n_tries, len(site_dists)))
615 alphas_gl1 = np.zeros((n_tries, len(site_dists)))
616 chi2_gl1 = np.zeros(n_tries)
617 # Minimización con múltiples ansatz
618 for a_gl1 in range(n_tries):
619   # Definición de ansatz
620   for value_gl1 in range(len(site_dists)):
621     ansatz_gl1[a_gl1][value_gl1] = random.uniform(-1,1)
622   # Minimización gradual
623   for turn in range(total_alphas):
624     # Coeficientes fijos
625     lower_alphas = np.zeros(total_alphas)
626     upper_alphas = np.zeros(total_alphas)
627     for n_alpha in range(total_alphas):
628       if n_alpha == turn:
629         lower_alphas[n_alpha] = -np.inf
630         upper_alphas[n_alpha] = np.inf
631       else:
632         lower_alphas[n_alpha] = alphas_gl1[a_gl1][n_alpha]
633         upper_alphas[n_alpha] = alphas_gl1[a_gl1][n_alpha]
634     bounds_alphas = Bounds(lower_alphas, upper_alphas)
635     # Minimización
636     solution_grad_l1 = minimize(min_cuadrados, alphas_gl1[a_gl1], method='
    trust-constr', constraints=linear_constraint, options={'disp': True})
637     alphas_gl1[a_gl1] = solution_grad_l1.x
638     chi2_gl1[a_gl1] = min_cuadrados(alphas_gl1[a_gl1])
639 # Almacenamiento
640 ansatz_gl1_fname = 'tesis_Egolf_EGb_Chi2Std_ProbE_L%(length)i_G%(grainsize)
    i_g%(coupling).3f_Niter%(iterations).3e_trans%(trans).3e_Nrondas%(turns)
    .3e_Nansatz%(ansatz)i_seed%(s)i_c%(ct).3f_sr%(ref)
    i_LeastSquares09_G_Local1_Ansatz.txt'
641 np.savetxt(ansatz_gl1_fname % dict_fname, ansatz_gl1)
642 alphas_gl1_fname = 'tesis_Egolf_EGb_Chi2Std_ProbE_L%(length)i_G%(grainsize)
```

```
    i_g%(coupling).3f_Niter%(iterations).3e_trans%(trans).3e_Nrondas%(turns)
    .3e_Nansatz%(ansatz)i_seed%(s)i_c%(ct).3f_sr%(ref)
    i_LeastSquares09_G_Local1_TotalAlphas.txt'
np.savetxt(alphas_gl1_fname % dict_fname, alphas_gl1)
chi2_gl1_fname = 'tesis_Egolf_EGb_Chi2Std_ProbE_L%(length)i_G%(grainsize)i_g
    %(coupling).3f_Niter%(iterations).3e_trans%(trans).3e_Nrondas%(turns).3
    e_Nansatz%(ansatz)i_seed%(s)i_c%(ct).3f_sr%(ref)
    i_LeastSquares09_G_Local1_Chi2.txt'
np.savetxt(chi2_gl1_fname % dict_fname, chi2_gl1)
# Identificación de chi2 mínima
min_chi2_gl1 = np.nanmin(chi2_gl1)
for counter_gl1, v_chi2_gl1 in enumerate(chi2_gl1):
  if v_chi2_gl1 == min_chi2_gl1:
    min_ansatz_gl1 = counter_gl1
# Alphas correspondiente a mínimo
alphas_grad_l1 = alphas_gl1[min_ansatz_gl1]
# Cálculo de probabilidades
pBoltz_grad_l1, Z_grad_l1 = prob_boltzmann(alphas_grad_l1)
Z_grad_l1 = np.array([Z_grad_l1])
px_grad_l1 = np.zeros(len(protoE))
for counter, prob in enumerate(pBoltz_grad_l1):
  px_grad_l1[counter] = prob*c
# Almacenamiento
fname_grad_l1 = 'tesis_Egolf_EGb_Chi2Std_ProbE_L%(length)i_G%(grainsize)i_g
    %(coupling).3f_Niter%(iterations).3e_trans%(trans).3e_Nrondas%(turns).3
    e_Nansatz%(ansatz)i_seed%(s)i_c%(ct).3f_sr%(ref)
    i_LeastSquares09_G_Local1_ProbBoltz.txt'
np.savetxt(fname_grad_l1 % dict_fname, pBoltz_grad_l1)
fname_Z_grad_l1 = 'tesis_Egolf_EGb_Chi2Std_ProbE_L%(length)i_G%(grainsize)
    i_g%(coupling).3f_Niter%(iterations).3e_trans%(trans).3e_Nrondas%(turns)
    .3e_Nansatz%(ansatz)i_seed%(s)i_c%(ct).3f_sr%(ref)
    i_LeastSquares09_G_Local1_Z.txt'
np.savetxt(fname_Z_grad_l1 % dict_fname, Z_grad_l1)
print('Diferencia con valores finales: ' + str(min_cuadrados(alphas_grad_l1)
    ))
print('Mínimos cuadrados (graduado - default) - FINAL')

### DICCIONARIO CON COEFICIENTES ALFA FINALES - GRADUADO L1
print('Diccionario con coeficientes alfa finales (graduado - \'trust-constr
    \') - INICIO')
dict_alphas_grad_l1 = {}
for n_distance in range(total_alphas):
  k_distance = list_key_dists[n_distance]
  dict_alphas_grad_l1[k_distance] = alphas_grad_l1[n_distance]
# Almacenamiento
fname_alphas_grad_l1 = 'tesis_Egolf_EGb_Chi2Std_ProbE_L%(length)i_G%(
    grainsize)i_g%(coupling).3f_Niter%(iterations).3e_trans%(trans).3
    e_Nrondas%(turns).3e_Nansatz%(ansatz)i_seed%(s)i_c%(ct).3f_sr%(ref)
    i_LeastSquares09_G_Local1_Alphas.txt'
with open(fname_alphas_grad_l1 % dict_fname, "w") as
    write_dict_alphas_grad_l1:
  json.dump(dict_alphas_grad_l1, write_dict_alphas_grad_l1)
print('Diccionario con coeficientes alfa finales (graduado - \'trust-constr
    \') - FINAL')

### ENERGÍAS FINALES - GRADUADO L1
print('Lista con Hamiltonianos finales (graduado - \'trust-constr\') -
    INICIO')
```

```
final_H_grad_l1 = np.zeros(len(protoE))
counter_H_grad_l1 = 0
for entry in protoE:
  H = 0
  for dist in entry[0]:
    term = dict_alphas_grad_l1[dist] * entry[0][dist]
    H = H + term
  final_H_grad_l1[counter_H_grad_l1] = H
  counter_H_grad_l1 = counter_H_grad_l1 + 1
# Almacenamiento
fname_H_grad_l1 = 'tesis_Egolf_EGb_Chi2Std_ProbE_L%(length)i_G%(grainsize)
    i_g%(coupling).3f_Niter%(iterations).3e_trans%(trans).3e_Nrondas%(turns)
    .3e_Nansatz%(ansatz)i_seed%(s)i_c%(ct).3f_sr%(ref)
    i_LeastSquares09_G_Local1_H.txt'
np.savetxt(fname_H_grad_l1 % dict_fname, final_H_grad_l1)
print('Lista con Hamiltonianos finales (graduado - \'trust-constr\') - FINAL
    ')
# Ordenamiento
final_H_grad_l1_gf, pBoltz_grad_l1_gf, prob_exp_grad_l1_gf = ordenador(
    final_H_grad_l1, pBoltz_grad_l1, prob_exp)
final_H_grad_l1_gf, px_grad_l1_gf, prob_exp_grad_l1_gf = ordenador(
    final_H_grad_l1, px_grad_l1, prob_exp)

### ESQUEMA ESTADO
es_gradl1, ps_gradl1, Zs_gradl1 = pe_estado(alphas_grad_l1)
Zs_gradl1 = np.array([Zs_gradl1])
# Almacenamiento
state_p_gradl1 = 'tesis_Egolf_EGb_Chi2Std_ProbE_L%(length)i_G%(grainsize)i_g
    %(coupling).3f_Niter%(iterations).3e_trans%(trans).3e_Nrondas%(turns).3
    e_Nansatz%(ansatz)i_seed%(s)i_c%(ct).3f_sr%(ref)
    i_LeastSquares09_G_Local1_ProbBoltzState.txt'
np.savetxt(state_p_gradl1 % dict_fname, ps_gradl1)
state_H_gradl1 = 'tesis_Egolf_EGb_Chi2Std_ProbE_L%(length)i_G%(grainsize)i_g
    %(coupling).3f_Niter%(iterations).3e_trans%(trans).3e_Nrondas%(turns).3
    e_Nansatz%(ansatz)i_seed%(s)i_c%(ct).3f_sr%(ref)
    i_LeastSquares09_G_Local1_HState.tyxt'
np.savetxt(state_H_gradl1 % dict_fname, es_gradl1)
state_Z_gradl1 = 'tesis_Egolf_EGb_Chi2Std_ProbE_L%(length)i_G%(grainsize)i_g
    %(coupling).3f_Niter%(iterations).3e_trans%(trans).3e_Nrondas%(turns).3
    e_Nansatz%(ansatz)i_seed%(s)i_c%(ct).3f_sr%(ref)
    i_LeastSquares09_G_Local1_ZState.tyxt'
np.savetxt(state_Z_gradl1 % dict_fname, Zs_gradl1)
# Ordenamiento
es_gradl1_gf, ps_gradl1_gf, hCG_gradl1_gf = ordenador(es_gradl1, ps_gradl1,
    hist_CG)

##############################################################################
##############################################################################

### GRÁFICA CON AJUSTE
plt.figure(2)
fig2, ((ax2a, ax2b),(ax2c,ax2d)) = plt.subplots(nrows=2, ncols=2, figsize
    =(27,30))
plt.rcParams.update({'font.size': 24})
plt.rcParams.update({'xtick.labelsize': 22})
plt.rcParams.update({'ytick.labelsize': 22})
plt.tight_layout(pad=4, h_pad=4, w_pad=4)
# Default
```

```
722 ax2a.set_title('(a) Local 0')
723 ax2a.set_yscale('log')
724 ax2a.set_ylabel(r'$P(H_{i})$', fontsize=22)
725 ax2a.set_xlabel(r'$H_{i}$',fontsize=22)
726 ax2a.tick_params(axis='both', which='major', labelsize=20)
727 ax2a.plot(final_H_def_gf, pBoltz_def_gf, 'r-', label = 'LS', linewidth=8)
728 ax2a.plot(final_H_def_gf, px_def_gf, 'g--', label = 'LS-R', markersize=18,
        linewidth=8)
729 ax2a.plot(final_H_def_gf, prob_exp_def_gf, 'bo', label = 'Exp',
        markerfacecolor='none', markeredgewidth=2, markersize=18)
730 ax2a.legend(loc='lower left', fontsize=22)
731 # Local 1
732 ax2b.set_title('(b) Local 1')
733 ax2b.set_yscale('log')
734 ax2b.set_ylabel(r'$P(H_{i})$',fontsize=22)
735 ax2b.set_xlabel(r'$H_{i}$',fontsize=22)
736 ax2b.tick_params(axis='both', which='major', labelsize=20)
737 ax2b.plot(final_H_l1_gf, pBoltz_l1_gf, 'r-', label = 'LS', linewidth=8)
738 ax2b.plot(final_H_l1_gf, px_l1_gf, 'g--', label = 'LS-R', markersize=18,
        linewidth=8)
739 ax2b.plot(final_H_l1_gf, prob_exp_l1_gf, 'bo', label = 'Exp',
        markerfacecolor='none', markeredgewidth=2, markersize=18)
740 ax2b.legend(loc='lower left', fontsize=22)
741 # Graduado - Default
742 ax2c.set_title('(c) Gradual 0')
743 ax2c.set_yscale('log')
744 ax2c.set_ylabel(r'$P(H_{i})$', fontsize=22)
745 ax2c.set_xlabel(r'$H_{i}$', fontsize=22)
746 ax2c.tick_params(axis='both', which='major', labelsize=20)
747 ax2c.plot(final_H_grad_def_gf, pBoltz_grad_def_gf, 'r-', label = 'LS',
        linewidth=8)
748 ax2c.plot(final_H_grad_def_gf, px_grad_def_gf, 'g--', label = 'LS-R',
        markersize=18, linewidth=8)
749 ax2c.plot(final_H_grad_def_gf, prob_exp_grad_def_gf, 'bo', label = 'Exp',
        markerfacecolor='none', markeredgewidth=2, markersize=18)
750 ax2c.legend(loc='lower left', fontsize=22)
751 # Graduado - Local 1
752 ax2d.set_title('(d) Gradual 1')
753 ax2d.set_yscale('log')
754 ax2d.set_ylabel(r'$P(H_{i})$', fontsize=22)
755 ax2d.set_xlabel(r'$H_{i}$', fontsize=22)
756 ax2d.tick_params(axis='both', which='major', labelsize=20)
757 ax2d.plot(final_H_grad_l1_gf, pBoltz_grad_l1_gf, 'r-', label = 'LS',
        linewidth=8)
758 ax2d.plot(final_H_grad_l1_gf, px_grad_l1_gf, 'g--', label = 'LS-R',
        markersize=18, linewidth=8)
759 ax2d.plot(final_H_grad_l1_gf, prob_exp_grad_l1_gf, 'bo', label = 'Exp',
        markerfacecolor='none', markeredgewidth=2, markersize=18)
760 ax2d.legend(loc='lower left', fontsize=22)
761 # Almacenamiento
762 imgname_Gibbs_graph = fname_H_l1 = 'tesis_Egolf_EGb_Chi2Std_ProbE_L%(length)
        i_G%(grainsize)i_g%(coupling).3f_Niter%(iterations).3e_trans%(trans).3
        e_Nrondas%(turns).3e_Nansatz%(ansatz)i_seed%(s)i_c%(ct).3f_sr%(ref)
        i_LeastSquares09_GraphTotal.png'
763 plt.savefig(imgname_Gibbs_graph % dict_fname)
764
765 ##############################################################################
766 ##############################################################################
```

```
767
768 ### GRÁFICA CON AJUSTE (Esquema Estado)
769 plt.figure(3)
770 fig3, ((ax3a, ax3b),(ax3c,ax3d)) = plt.subplots(nrows=2, ncols=2, figsize
    =(27,30))
771 plt.rcParams.update({'font.size': 24})
772 plt.rcParams.update({'xtick.labelsize': 22})
773 plt.rcParams.update({'ytick.labelsize': 22})
774 plt.tight_layout(pad=4, h_pad=4, w_pad=4)
775 # Default
776 #es_gradl1_gf, ps_gradl1_gf, hCG_gradl1_gf
777 ax3a.set_title('(a) Local 0')
778 ax3a.set_yscale('log')
779 ax3a.set_ylabel(r'$P(S_{i})$', fontsize=22)
780 ax3a.set_xlabel(r'$H(S_{i})$',fontsize=22)
781 ax3a.tick_params(axis='both', which='major', labelsize=20)
782 ax3a.plot(es_def_gf, ps_def_gf, 'r-', label = 'LS', linewidth=8)
783 ax3a.plot(es_def_gf, hCG_def_gf, 'bo', label = 'Exp', markerfacecolor='none'
    , markeredgewidth=2, markersize=18)
784 ax3a.legend(loc='lower left', fontsize=22)
785 # Local 1
786 ax3b.set_title('(b) Local 1')
787 ax3b.set_yscale('log')
788 ax3b.set_ylabel(r'$P(H_{i})$',fontsize=22)
789 ax3b.set_xlabel(r'$H_{i}$',fontsize=22)
790 ax3b.tick_params(axis='both', which='major', labelsize=20)
791 ax3b.plot(es_l1_gf, ps_l1_gf, 'r-', label = 'LS', linewidth=8)
792 ax3b.plot(es_l1_gf, hCG_l1_gf, 'bo', label = 'Exp', markerfacecolor='none',
    markeredgewidth=2, markersize=18)
793 ax3b.legend(loc='lower left', fontsize=22)
794 # Graduado - Default
795 ax3c.set_title('(c) Gradual 0')
796 ax3c.set_yscale('log')
797 ax3c.set_ylabel(r'$P(H_{i})$', fontsize=22)
798 ax3c.set_xlabel(r'$H_{i}$', fontsize=22)
799 ax3c.tick_params(axis='both', which='major', labelsize=20)
800 ax3c.plot(es_graddef_gf, ps_graddef_gf, 'r-', label = 'LS', linewidth=8)
801 ax3c.plot(es_graddef_gf, hCG_graddef_gf, 'bo', label = 'Exp',
    markerfacecolor='none', markeredgewidth=2, markersize=18)
802 ax3c.legend(loc='lower left', fontsize=22)
803 # Graduado - Local 1
804 ax3d.set_title('(d) Gradual 1')
805 ax3d.set_yscale('log')
806 ax3d.set_ylabel(r'$P(H_{i})$', fontsize=22)
807 ax3d.set_xlabel(r'$H_{i}$', fontsize=22)
808 ax3d.tick_params(axis='both', which='major', labelsize=20)
809 ax3d.plot(es_gradl1_gf, ps_gradl1_gf, 'r-', label = 'LS', linewidth=8)
810 ax3d.plot(es_gradl1_gf, hCG_gradl1_gf, 'bo', label = 'Exp', markerfacecolor=
    'none', markeredgewidth=2, markersize=18)
811 ax3d.legend(loc='lower left', fontsize=22)
812 # Almacenamiento
813 imgname_Gis_graph = 'tesis_Egolf_EGb_Chi2Std_ProbE_L%(length)i_G%(grainsize)
    i_g%(coupling).3f_Niter%(iterations).3e_trans%(trans).3e_Nrondas%(turns)
    .3e_Nansatz%(ansatz)i_seed%(s)i_c%(ct).3f_sr%(ref)
    i_LeastSquares09_GraphTotalState.png'
814 plt.savefig(imgname_Gis_graph % dict_fname)
815
816 ##############################################################################
```

```
817 ##############################################################################
818
819 ### GRÁFICA CON AJUSTE BAJO ESQUEMA GRADUADO DEFAULT
820 plt.figure(2)
821 fig2, ax3 = plt.subplots(nrows=1, ncols=1, figsize=(27,30))
822 plt.rcParams.update({'font.size': 24})
823 plt.rcParams.update({'xtick.labelsize': 22})
824 plt.rcParams.update({'ytick.labelsize': 22})
825 plt.tight_layout(pad=4, h_pad=4, w_pad=4)
826 # Graduado - Default
827 ax3.set_title('(c) Gradual 0')
828 ax3.set_yscale('log')
829 ax3.set_ylabel(r'$P(H_{i})$', fontsize=22)
830 ax3.set_xlabel(r'$H_{i}$', fontsize=22)
831 ax3.tick_params(axis='both', which='major', labelsize=20)
832 ax3.plot(final_H_grad_def_gf, pBoltz_grad_def_gf, 'r-', label = 'LS',
       linewidth=8)
833 ax3.plot(final_H_grad_def_gf, px_grad_def_gf, 'g--', label = 'LS-R',
       markersize=18, linewidth=8)
834 ax3.plot(final_H_grad_def_gf, prob_exp_grad_def_gf, 'bo', label = 'Exp',
       markerfacecolor='none', markeredgewidth=2, markersize=18)
835 ax3.legend(loc='lower left', fontsize=22)
836 # Almacenamiento
837 imgname_Gibbs_graph = fname_H_l1 = 'tesis_Egolf_EGb_Chi2Std_ProbE_L%(length)
       i_G%(grainsize)i_g%(coupling).3f_Niter%(iterations).3e_trans%(trans).3
       e_Nrondas%(turns).3e_Nansatz%(ansatz)i_seed%(s)i_c%(ct).3f_sr%(ref)
       i_LeastSquares09_Graph_G0.png'
838 plt.savefig(imgname_Gibbs_graph % dict_fname)
839
840 print('Programa concluido')
```

Apéndice K

Código para pruebas de balance detallado y distribución de Boltzmann en modelo de Ising

```
import sys
import numpy as np
import networkx as nx
import random
import matplotlib.pyplot as plt
import scipy
from scipy.optimize import curve_fit
from scipy.optimize import minimize_scalar
from scipy.optimize import minimize
from scipy.optimize import Bounds
from scipy.optimize import LinearConstraint
import json

##############################################################################
##############################################################################

# Cálculo de distancias entre elementos en una rejilla cuadrada
    bidimensional bajo condiciones de frontera periódicas
def calc_dist(loc1, loc2, longitud):
  distx = loc1[0] - loc2[0]
  disty = loc1[1] - loc2[1]
  dist1 = (distx ** 2 + disty ** 2) ** (1 / 2.0)
  dist2 = ((distx - longitud) ** 2 + disty ** 2) ** (1 / 2.0)
  dist3 = ((distx + longitud) ** 2 + disty ** 2) ** (1 / 2.0)
  dist4 = (distx ** 2 + (disty - longitud) ** 2) ** (1 / 2.0)
  dist5 = (distx ** 2 + (disty + longitud) ** 2) ** (1 / 2.0)
  dist6 = ((distx + longitud) ** 2 + (disty + longitud) ** 2) ** (1 / 2.0)
  dist7 = ((distx - longitud) ** 2 + (disty + longitud) ** 2) ** (1 / 2.0)
  dist8 = ((distx - longitud) ** 2 + (disty - longitud) ** 2) ** (1 / 2.0)
  dist9 = ((distx + longitud) ** 2 + (disty - longitud) ** 2) ** (1 / 2.0)
  lista_distancias = [dist1, dist2, dist3, dist4, dist5, dist6, dist7, dist8
    , dist9]
  distancia = min(lista_distancias)
  return distancia

# Iniciación de gráfica para rejilla CML a grano grueso en estado 0
def ini_graf(longitud):
```

```
36   grafica = nx.Graph()
37   num_sitios = longitud**2
38   # Definición de nodos
39   for sitio in range(num_sitios):
40     grafica.add_node(sitio, u = -1)
41   # Definición de edges
42   lista_edges = []
43   for sitio1 in range(num_sitios):
44     for sitio2 in range(num_sitios):
45       if sitio2 == sitio1:
46         continue
47       coord1 = (sitio1 % longitud, sitio1 // longitud)
48       coord2 = (sitio2 % longitud, sitio2 // longitud)
49       distancia = calc_dist(coord1, coord2, longitud)
50       lista_edges.append((sitio1, sitio2, {'path_distance': distancia}))
51   grafica.add_edges_from(lista_edges)
52   return grafica
53
54 ### CÁLCULO DE PRIMEROS VECINOS PARA UN NODO PARTICULAR DE LA GRÁFICA
55 def primeros_vecinos(graf,nodo):
56   lista_dist = []
57   for vec1, datos1 in graf.adj[nodo].items():
58     for keys1, dists1 in datos1.items():
59       lista_dist.append(dists1)
60   dist_min = min(lista_dist)
61   lista_1v = []
62   for vec2, datos2 in graf.adj[nodo].items():
63     for keys2, dists2 in datos2.items():
64       if dists2 == dist_min:
65         lista_1v.append(vec2)
66   return dist_min, lista_1v
67
68 # Configuración de gráfica en un estado definido
69 def def_edo(estado, grafica):
70   referencia = estado
71   exp = 0
72   while referencia > 0:
73     factor2 = 2 ** (exp)
74     tasa = referencia / factor2
75     if tasa >= 1:
76       exp = exp + 1
77     else:
78       grafica.nodes[exp - 1]['u'] = 1
79       factorneg = 2 ** (exp - 1)
80       referencia = referencia - factorneg
81       exp = 0
82   return grafica
83
84 # Identificación de estado y energía de rejilla de Ising
85 def id_edo_hi(grafica, lista_1vecinos, acoplamiento):
86   # Identificación de estado
87   suma_id = 0
88   lista_sitios = list(grafica.nodes())
89   for sitio in lista_sitios:
90     if grafica.nodes[sitio]['u'] == -1:
91       continue
92     elif grafica.nodes[sitio]['u'] == 1:
93       termino = 2 ** sitio
```

```
94        suma_id = suma_id + termino
95    # Identificación de energía correspondiente
96    # Cálculo de productos entre sitio particular y primeros vecinos
97    list_prod = []
98    total_sitios = len(list(grafica.nodes()))
99    for sitio1 in range(total_sitios):
100     for sitio2 in lista_1vecinos[sitio1][1]:
101       if sitio2 <= sitio1:
102         continue
103       producto = grafica.nodes[sitio1]['u'] * grafica.nodes[sitio2]['u']
104       list_prod.append(producto)
105   # Cálculo de energía como producto entre coeficiente de acoplamiento y
      suma de productos
106   suma_prod = sum(list_prod)
107   energia = -acoplamiento * suma_prod
108   return suma_id, energia
109
110 # Actualización de rejilla con algoritmo de Metropolis
111 def update_metropolis(grafica, lista_1vecinos, acoplamiento):
112   # Cálculo del número total de sitios
113   total_sitios = len(list(grafica.nodes))
114   # Identificación de estado y energía actuales
115   estado_actual, energia_actual = id_edo_hi(grafica, lista_1vecinos,
      acoplamiento)
116   # Cambio de espín aleatorio
117   sitio_aleatorio = random.randint(0, total_sitios - 1)
118   grafica.nodes[sitio_aleatorio]['u'] = -grafica.nodes[sitio_aleatorio]['u']
119   # Identificación de estado y energía tras cambio de espín
120   estado_modif, energia_modif = id_edo_hi(grafica, lista_1vecinos,
      acoplamiento)
121   # Cálculo de cambio de energía
122   diferencia_energia = energia_modif - energia_actual
123   prob_trans = np.exp(-diferencia_energia)
124   ref_aleatoria = random.random()
125   # Casos posibles
126   if diferencia_energia <= 0:
127     return grafica
128   elif ref_aleatoria < prob_trans:
129     return grafica
130   else:
131     grafica.nodes[sitio_aleatorio]['u'] = -grafica.nodes[sitio_aleatorio]['u
      ']
132     return grafica
133
134 def minPos(probabilidades):
135     probs_pos = []
136     for prob in probabilidades:
137         if prob > 0:
138             probs_pos.append(prob)
139     val = min(probs_pos)
140     return val
141
142 def id_pos0(probabilidades):
143     counter_0
144     for c_prob, prob in enumerate(probabilidades):
145         if prob == 0.0:
146             counter_0 = c_prob
147             break
```

```
148     return counter_0
149 
150 ##################### Ajustes inclusivos ########################
151 
152 def probabilidad_Boltzmann(energias, b):                                    #
    ######################### curve_fit ############################
153     probabilidades = np.zeros(len(energias))
154     Z = 0
155     for counter_e, energia in enumerate(energias):
156         # Definición (gradual) de función de partición
157         Z = Z + protoE[counter_e][2] * np.exp(b * energia)
158         # Definición (parcial) de probabilidad
159         probabilidades[counter_e] = protoE[counter_e][2] * np.exp(b *
    energia)
160     probabilidades = probabilidades / Z
161     return probabilidades
162 
163 def prob_Boltzmann_ms(b):                                                   #
    ######################## minimize_scalar ####################
164     pB_ms = np.zeros(len(protoE))
165     Z = 0
166     for counter_ms, entry_ms in enumerate(protoE):
167         # Definición (gradual) de función de partición
168         Z = Z + entry_ms[2] * np.exp(b * entry_ms[0])
169         # Definición (parcial) de probabilidad
170         pB_ms[counter_ms] = entry_ms[2] * np.exp(b * entry_ms[0])
171     pB_ms = pB_ms / Z
172     return pB_ms
173 
174 def res2_ms(b):
175     # Cálculo de probabilidades de Boltzmann
176     pms_calc = prob_Boltzmann_ms(b)
177     # Cálculo de suma de cuadrados de residuos
178     sum_res2_ms = 0
179     for counter_ms, pv_ms in enumerate(probx_values):
180         sum_res2_ms = sum_res2_ms + (pv_ms - pms_calc[counter_ms]) ** 2
181     return sum_res2_ms
182 
183 def prob_Boltzmann_min(b):                                                  #
    ########################## minimize ##########################
184     pB_min = np.zeros(len(protoE))
185     Z = 0
186     for counter_ms, entry_ms in enumerate(protoE):
187         # Definición (gradual) de función de partición
188         Z = Z + entry_ms[2] * np.exp(b[0] * entry_ms[0])
189         # Definición (parcial) de probabilidad
190         pB_min[counter_ms] = entry_ms[2] * np.exp(b[0] * entry_ms[0])
191     pB_min = pB_min / Z
192     return pB_min
193 
194 def res2_min(b):
195     # Cálculo de probabilidades de Boltzmann
196     pms_calc = prob_Boltzmann_min(b)
197     # Cálculo de suma de cuadrados de residuos
198     sum_res2_ms = 0
199     for counter_ms, pv_ms in enumerate(probx_values):
200         sum_res2_ms = sum_res2_ms + (pv_ms - pms_calc[counter_ms]) ** 2
201 #         print(counter_ms, (pv_ms - pms_calc[counter_ms]) ** 2)
```

```
202 #     print('########### chi2: ' + str(sum_res2_ms))
203     return sum_res2_ms
204 
205 ##############################################################################
206 
207 ### PROBABILIDADES DE BOLTZMANN PARA ESTADOS
208 def pe_estado(b):
209     p_edo = np.zeros(N_total_states)
210     e_edo = np.zeros(N_total_states)
211     funcion_Z = 0
212     for estado in range(N_total_states):
213         clave = 's' + str(float(estado))
214         # Definición (gradual) de función de partición
215         funcion_Z = funcion_Z + np.exp(b[0] * dict_sums[clave])
216         # Definición (parcial) de probabilidad
217         p_edo[estado] = np.exp(b[0] * dict_sums[clave])
218         # Definición de energía
219         e_edo[estado] = -b[0] * dict_sums[clave]
220     p_edo = p_edo / funcion_Z
221     return p_edo, e_edo, funcion_Z
222 
223 ### ORDENADOR
224 def ordenador(energias_ajuste, probabilidades_ajuste,
        probabilidades_experimentales):
225   # Ordenamiento de energías resultantes de ajuste
226   energias_ordenadas = sorted(energias_ajuste)
227   # Identificación de índices
228   indices = []
229   for energia_ord in energias_ordenadas:
230     for ind, energia in enumerate(energias_ajuste):
231       if energia == energia_ord:
232         indices.append(ind)
233         break
234   # Ordenamiento de probabilidades
235   prob_ajuste_ordenadas = []
236   prob_exp_ordenadas = []
237   for ind in indices:
238     prob_ajuste_ordenadas.append(probabilidades_ajuste[ind])
239     prob_exp_ordenadas.append(probabilidades_experimentales[ind])
240   return energias_ordenadas, prob_ajuste_ordenadas, prob_exp_ordenadas
241 
242 ##############################################################################
243 ##############################################################################
244 
245 ### CONFIGURACIÓN DE REJILLA DE ISING
246 # Solicitud de parámetros de rejilla de Ising
247 s = random.randrange(sys.maxsize)
248 random.seed(s)
249 L = int(input('Ingresa la longitud de la rejilla (entero): '))
250 N_total_sites = L ** 2
251 J = float(input('Ingresa el valor del acoplamiento entre primeros vecinos (
        real): '))
252 Jguess = float(input('Ingresa ansatz para acoplamiento entre primeros
        vecinos (real): '))
253 N_turns = int(input('Ingresa el total de rondas por sitio (entero): '))
254 c = float(input('Ingresa desplazamiento (real): '))
255 file_dict = {'length': L, 'coupling': J, 'guess': Jguess, 'turns': N_turns,
        'seed': s, 'cte': c}
```

```
# Definición de rejilla
lattice_ising = ini_graf(L)
list_1n_ising = []
for site_ising in range(int(L ** 2)):
  dist_ising, list1v_ising = primeros_vecinos(lattice_ising, site_ising)
  list_1n_ising.append((site_ising,list1v_ising))

### EVOLUCIÓN TEMPORAL DE SISTEMA ("FINO" Y A GRANO GRUESO)
N_total_states = 2 ** N_total_sites
counter_zero = 0
# Configuración de gráfica en un estado inicial aleatorio
first_state = random.randint(0,N_total_states)
lattice_ising = def_edo(first_state, lattice_ising)
# Definición de archivo con evolución temporal
fname_metropolis = 'tesis_IsingTest_scBD_L%(length)i_J%(coupling).3f_Jguess
    %(guess).3f_Nrondas%(turns).3e_seed%(seed)i_ListOfStates_Ising.txt'
# Parámetros adicionales
metropolis = []
total_iterations = N_turns * N_total_sites
list_CG_states = np.zeros(total_iterations)
tenth_progress = int(total_iterations / 10)

for loop in range(total_iterations):
  # Marcador de progreso
  ind_progress = loop % tenth_progress
  if ind_progress == 0:
    print('.')
  # Update de rejilla CML
  lattice = update_metropolis(lattice_ising, list_1n_ising, J)
  lattice_state, lattice_energy = id_edo_hi(lattice_ising, list_1n_ising, J)
  metropolis.append(lattice_state)
# Almacenamiento de datos
metropolis = np.array(metropolis)
np.savetxt(fname_metropolis % file_dict, metropolis)

### BALANCE DETALLADO IIIA - MATRIZ DE TRANSICIONES PARA REJILLA "FINA"
matrix_transitions_metropolis = np.zeros((2**(N_total_sites),2**(
    N_total_sites)))
fine_total_iterations = len(metropolis)
for state in range(1,fine_total_iterations):
  matrix_row_metropolis = int(metropolis[state-1])
  matrix_column_metropolis = int(metropolis[state])
  matrix_transitions_metropolis[matrix_row_metropolis][
    matrix_column_metropolis] = matrix_transitions_metropolis[
    matrix_row_metropolis][matrix_column_metropolis] + 1
print("Matriz de transiciones definida")

### BALANCE DETALLADO IVA - DICCIONARIO PARA GRAFICAR PROMEDIOS <(Delta/N)
    ^2> PARA REJILLA "FINA"
fname_dictorig_metro = "tesis_IsingTest_scBD_L%(length)i_J%(coupling).3
    f_Jguess%(guess).3f_Nrondas%(turns).3e_seed%(seed)i_Ising_DictOriginal.
    txt"
dict_db_metro = {}
for row in range(2**(N_total_sites) - 1):
  for column in range(row + 1, 2**(N_total_sites)):
    pos_trans_metro = matrix_transitions_metropolis[row][column]
    neg_trans_metro = matrix_transitions_metropolis[column][row]
    delta_metro = pos_trans_metro - neg_trans_metro
```

```
    total_metro = pos_trans_metro + neg_trans_metro
    if total_metro == 0:
      continue
    key_total_metro = 'a' + str(total_metro)
    if key_total_metro in dict_db_metro:
      dict_db_metro[key_total_metro][0] = dict_db_metro[key_total_metro][0]
    + delta_metro ** 2
      dict_db_metro[key_total_metro][1] = dict_db_metro[key_total_metro][1]
    + 1
    else:
      dict_db_metro[key_total_metro] = [delta_metro ** 2, 1]
# Almacenamiento
with open(fname_dictorig_metro % file_dict, "w") as write_dicto_metro:
  json.dump(dict_db_metro, write_dicto_metro)
print("Diccionario 1 definido")

### BALANCE DETALLADO V - GRÁFICA DE PROMEDIOS <(Delta/N)^2>
# Arreglos para rejilla "fina"
M = len(dict_db_metro)
metro_N = np.zeros(M)
metro_avg = np.zeros(M)
# Cálculo de promedios para rejilla "fina"
counter_entry_dbo_metro = 0
for key in dict_db_metro:
  valueN_metro = int(float(key[1:]))
  metro_N[counter_entry_dbo_metro] = valueN_metro
  n_entries_metro = dict_db_metro[key][1]
  ## "A MANO" ##
  # Promedio "a mano"
  sum_avg_metro = dict_db_metro[key][0] / (valueN_metro ** 2)
  avg_metro = sum_avg_metro / n_entries_metro
  metro_avg[counter_entry_dbo_metro] = avg_metro
  counter_entry_dbo_metro = counter_entry_dbo_metro + 1

# Cálculo de chi^2
chi2 = 0
for counter_N, vN in enumerate(metro_N):
  theoretical = 1.0 / vN
  diff = (metro_avg[counter_N] - theoretical) ** 2
  chi2 = chi2 + diff

max_N = max(metro_N)
x_ref = np.arange(1,max_N,0.1)
y_ref = np.zeros(len(x_ref))
for x in range(len(x_ref)):
  y_ref[x] = 1 / x_ref[x]

figC2, axC2 = plt.subplots(nrows=1, ncols=1, figsize=(10,10))
plt.rcParams.update({'font.size': 24})
plt.rcParams.update({'xtick.labelsize': 22})
plt.rcParams.update({'ytick.labelsize': 22})
plt.tight_layout(pad=4, h_pad=4, w_pad=4)

axC2.set_title('Balance Detallado (Ising) - %(m)i valores de ' % {'m': M} +
    r'$N$' + ', ' + r'$ \chi^{2} = %(dif).3e $' % {'dif': chi2}, size=18)
axC2.set_yscale('log')
axC2.set_ylabel(r'$\left\langle \left( \Delta / N \right)^{2} \right\rangle$
    ', fontsize=22)
```

```
axC2.set_xscale('log')
axC2.set_xlabel(r'$N$', fontsize=22)
axC2.plot(x_ref, y_ref, 'r-', label=r'$N^{-1}$', markersize=18, linewidth=8)
axC2.plot(metro_N, metro_avg, 'bo', label='Metropolis', markerfacecolor='
    none', markeredgewidth=2, markersize=18)
axC2.legend(loc='lower left', fontsize='small')

plt.savefig("tesis_IsingTest_scBD_L%(length)i_J%(coupling).3f_Jguess%(guess)
    .3f_Nrondas%(turns).3e_seed%(seed)i_Ising_Graph.png" % file_dict)


### DISTRIBUCIÓN DE PROBABILIDAD DADA LA EVOLUCIÓN DE LA REJILLA DE ISING
hist_metro, bins_metro = np.histogram(metropolis, bins = N_total_states,
    range = (0, N_total_states), density = True)
# Almacenamiento de histograma (probabilidades "experimentales")
hist_metro_fname = 'tesis_IsingTest_scBD_L%(length)i_J%(coupling).3f_Jguess
    %(guess).3f_Nrondas%(turns).3e_seed%(seed)i_c%(cte).3f_HistDB.txt'
np.savetxt(hist_metro_fname % file_dict, hist_metro)

################################################################################
################################################################################

### DICCIONARIO CON ESTADOS S_I Y SUMAS DE PRODUCTOS ENTRE PARES DE ESPÍN
lattice_ising = ini_graf(L)
# Diccionario con sumas de energías
dict_sums = {}
tenth_progress = int(N_total_states / 10)
for state in range(N_total_states):
  # Marcador de progreso
  if state % tenth_progress == 0:
    print('*')
  # Definición de gráfica en estado definido
  lattice_ising = def_edo(state,lattice_ising)
  # Cálculo de productos
  list_prod = []
  for site1 in range(int(L ** 2)):
    for site2 in list_1n_ising[site1][1]:
      if site2 <= site1:
        continue
      product = lattice_ising.nodes[site1]['u'] * lattice_ising.nodes[site2
    ]['u']
      list_prod.append(product)
  sum_prod = sum(list_prod)
  energy = sum_prod
  # Almacenamiento de energía
  key_state = 's' + str(float(state))
  dict_sums[key_state] = energy
  lattice_ising = ini_graf(L)
# Almacenamiento de diccionario
dict_sumE_fname = 'tesis_IsingTest_scEG_L%(length)i_J%(coupling).3f_Jguess%(
    guess).3f_Nrondas%(turns).3e_seed%(seed)i_c%(cte).3f_ProtoE_dict.txt'
with open(dict_sumE_fname % file_dict, "w") as write_dictsums:
  json.dump(dict_sums, write_dictsums)

### LISTA CON SUMAS DE PRODUCTOS ENTRE PARES DE ESPÍN (PROTO-ENERGÍAS)
list_sums = []
for sum_prod in dict_sums.values():
  if sum_prod not in list_sums:
```

```
413     list_sums.append(sum_prod)
414   else:
415     continue
416 # Almacenamiento de lista con sumas
417 listsums_fname = 'tesis_IsingTest_scEG_L%(length)i_J%(coupling).3f_Jguess%(
      guess).3f_Nrondas%(turns).3e_seed%(seed)i_c%(cte).3f_ProtoE_list.txt'
418 with open(listsums_fname % file_dict, "w") as write_listsums:
419   json.dump(list_sums, write_listsums)
420
421 ### LISTA CON PROTO-ENERGÍAS, PROBABILIDADES Y NÚMERO DE ESTADOS
      CORRESPONDIENTES
422 protoE_desord = []
423 for sum_prod in list_sums:
424   index_holder = []
425   for key_sum in dict_sums:
426     if dict_sums[key_sum] == sum_prod:
427       state = int(float(key_sum[1:]))
428       index_holder.append(state)
429     else:
430       continue
431   num_states_H = len(index_holder)
432   sum_probs = 0
433   for index in index_holder:
434     term = hist_metro[index]
435     sum_probs = sum_probs + term
436   protoE_desord.append([sum_prod, sum_probs, num_states_H])
437 # Almacenamiento de lista con probabilidades y total de estados
      correspondientes a proto-energías
438 pE_desord_fname = 'tesis_IsingTest_scEG_L%(length)i_J%(coupling).3f_Jguess%(
      guess).3f_Nrondas%(turns).3e_seed%(seed)i_c%(cte).3f_ProtoE&Prob&
      NS_desordenadas.txt'
439 with open(pE_desord_fname % file_dict, "w") as write_protoE_desord:
440   json.dump(protoE_desord, write_protoE_desord)
441
442 ### ORDENAMIENTO DE PROTO-ENERGÍAS POR PROBABILIDAD
443 # Lista con probabilidades de proto-energías, ordenadas de mayor a menor
444 prob_ord = []
445 for pair in protoE_desord:
446   prob_ord.append(pair[1])
447 prob_ord = sorted(prob_ord, reverse = True)
448 # Ordenamiento de lista con (proto-)energías y probabilidades asociadas
449 finalE = []
450 protoE = []
451 for probability in prob_ord:
452   for entry in protoE_desord:
453     if entry[1] == probability:
454       if entry not in protoE:
455         energ = (-J*entry[0])
456         finalE.append([energ,entry[1],entry[2]])
457         protoE.append(entry)
458         break
459       else:
460         continue
461     else:
462       continue
463 # Almacenamiento de lista con probabilidades y total de estados
      correspondientes a proto-energías
464 finalE_fname = 'tesis_IsingTest_scEG_L%(length)i_J%(coupling).3f_Jguess%(
```

```
    guess).3f_Nrondas%(turns).3e_seed%(seed)i_c%(cte).3f_E&Prob&NS.txt'
with open(finalE_fname % file_dict, "w") as write_finalE:
  json.dump(finalE, write_finalE)
protoE_fname = 'tesis_IsingTest_scEG_L%(length)i_J%(coupling).3f_Jguess%(
    guess).3f_Nrondas%(turns).3e_seed%(seed)i_c%(cte).3f_ProtoE&Prob&NS.txt'
with open(protoE_fname % file_dict, "w") as write_protoE:
  json.dump(protoE, write_protoE)

################################################################################
################################################################################

### ARREGLOS CON VALORES RELEVANTES
# Proto-energías
pE_values = np.array([pe[0] for pe in protoE])
# Energías
E_values = np.array([ev[0] for ev in finalE])
# Probabilidades
probx_values = np.array([pe[1] for pe in protoE])

################################################################################
################################################################################
######################## Ajustes inclusivos ####################################

### AJUSTE (curve_fit)
print('############################### curve_fit
    ##########################################')
# Ajuste
J_cf, cov_cf = curve_fit(probabilidad_Boltzmann, pE_values, probx_values, p0
    =Jguess)
# Almacenamiento
Jcf_fname = 'tesis_IsingTest_scEG_chi2_std_L%(length)i_J%(coupling).3
    f_Jguess%(guess).3f_Nrondas%(turns).3e_seed%(seed)i_c%(cte).3
    f_LS_CurveFit_J.txt'
np.savetxt(Jcf_fname % file_dict, J_cf)
# Esquema Estado
ps_cf, es_cf, Zs_cf = pe_estado(J_cf)

# Residuos
chi2_cf = np.sum((np.array(ps_cf) - np.array(hist_metro)) ** 2)
# Almacenamiento
chi2cf_fname = 'tesis_IsingTest_scEG_chi2_std_L%(length)i_J%(coupling).3
    f_Jguess%(guess).3f_Nrondas%(turns).3e_seed%(seed)i_c%(cte).3
    f_LS_CurveFit_chi2.txt'
np.savetxt(chi2cf_fname % file_dict, np.array([chi2_cf]))


### AJUSTE (minimize_scalar)
print('############################### minimize_scalar
    ##########################################')
# Ajuste
res_ms = minimize_scalar(res2_ms)
J_ms_v = res_ms.x
J_ms = np.array([J_ms_v])
# Almacenamiento
Jms_fname = 'tesis_IsingTest_scEG_chi2_std_L%(length)i_J%(coupling).3
    f_Jguess%(guess).3f_Nrondas%(turns).3e_seed%(seed)i_c%(cte).3
    f_LS_MinimizeScalar_J.txt'
np.savetxt(Jms_fname % file_dict, J_ms)
```

```
512 # Esquema Estado
513 ps_ms, es_ms, Zs_ms = pe_estado(J_ms)
514
515 # Residuos
516 chi2_ms = np.sum((np.array(ps_ms) - np.array(hist_metro)) ** 2)
517 # Almacenamiento
518 chi2ms_fname = 'tesis_IsingTest_scEG_chi2_std_L%(length)i_J%(coupling).3
    f_Jguess%(guess).3f_Nrondas%(turns).3e_seed%(seed)i_c%(cte).3
    f_LS_MinimizeScalar_chi2.txt'
519 np.savetxt(chi2ms_fname % file_dict, np.array([chi2_ms]))
520
521 ### AJUSTE (minimize)
522 print('##################################### minimize
    ##########################################')
523 # Ajuste
524 res_min = minimize(res2_min, Jguess)
525 J_min = res_min.x
526 # Almacenamiento
527 Jmin_fname = 'tesis_IsingTest_scEG_chi2_std_L%(length)i_J%(coupling).3
    f_Jguess%(guess).3f_Nrondas%(turns).3e_seed%(seed)i_c%(cte).3
    f_LS_Minimize_J.txt'
528 np.savetxt(Jmin_fname % file_dict, J_min)
529 # Esquema Estado
530 ps_min, es_min, Zs_min = pe_estado(J_min)
531
532
533 # Residuos
534 chi2_min = np.sum((np.array(ps_min) - np.array(hist_metro)) ** 2)
535 # Almacenamiento
536 chi2min_fname = 'tesis_IsingTest_scEG_chi2_std_L%(length)i_J%(coupling).3
    f_Jguess%(guess).3f_Nrondas%(turns).3e_seed%(seed)i_c%(cte).3
    f_LS_Minimize_chi2.txt'
537 np.savetxt(chi2min_fname % file_dict, np.array([chi2_min]))
538
539
540 # Proto-energías
541 print('Proto-energías: ')
542 print(pE_values)
543 # Probabilidades
544 print('Probabilidades: ')
545 print(probx_values)
546 # Energías
547 print('Energías (T): ')
548 print(E_values)
549 print('Energías (fit inclusivo - curve_fit): ')
550 fitE_values_cf = np.array(-J_cf*pE_values)
551 print(fitE_values_cf)
552
553 print('Energías (fit  inclusivo - minimize_scalar): ')
554 fitE_values_ms = np.array(-J_ms_v*pE_values)
555 print(fitE_values_ms)
556
557 print('Energías (fit  inclusivo - minimize): ')
558 fitE_values_min = np.array(-J_min[0]*pE_values)
559 print(fitE_values_min)
560
561
562 ### CURVA TEÓRICA
```

```
ps_teo, es_teo, Zs_teo = pe_estado(np.array([J]))

# Residuos
chi2_teo = np.sum((np.array(ps_teo) - np.array(hist_metro)) ** 2)
# Almacenamiento
chi2t_fname = 'tesis_IsingTest_scEG_chi2_std_L%(length)i_J%(coupling).3
    f_Jguess%(guess).3f_Nrondas%(turns).3e_seed%(seed)i_c%(cte).3
    f_LS_Teorico_chi2.txt'
np.savetxt(chi2t_fname % file_dict, np.array([chi2_teo]))


################################################################################
################################################################################

### GRÁFICA CON AJUSTE (Curve Fit)
plt.figure(1)
#fig3, ((ax3a, ax3b), (ax3c, ax3d), (ax3e, ax3f), (ax3g, ax3h)) = plt.
    subplots(nrows=4, ncols=2, figsize=(30,60))
fig1, ax1 = plt.subplots(nrows=1, ncols=1, figsize=(10,10))
#fig3, ax3a = plt.subplots(nrows=1, ncols=1, figsize=(18,18))
plt.rcParams.update({'font.size': 24})
plt.rcParams.update({'xtick.labelsize': 22})
plt.rcParams.update({'ytick.labelsize': 22})
plt.tight_layout(pad=4, h_pad=4, w_pad=4)
####################
#### Inclusivo ####
####################
ax1.set_title('Ising ' + r'$(J=%5.3f, J_{g}=%5.3f)$' % tuple([J,Jguess]))
ax1.set_yscale('log')
ax1.set_ylabel(r'$P(S_{i})$', fontsize=22)
max_prob = max(hist_metro)
min_prob = minPos(hist_metro)
ax1.set_ylim(bottom=min_prob/10.0, top=max_prob*10)
ax1.set_xlabel(r'$H(S_{i})$',fontsize=22)
minE_inc = min(min(fitE_values_cf), min(fitE_values_ms), min(fitE_values_min
    ), min(E_values))
maxE_inc = max(max(fitE_values_cf), max(fitE_values_ms), min(fitE_values_min
    ), max(E_values))
delE_inc = (maxE_inc - minE_inc) / 10.0
ax1.set_xlim(minE_inc - delE_inc, maxE_inc + delE_inc)
ax1.tick_params(axis='both', which='major', labelsize=20)
# Teóricos
ax1.plot(es_teo, hist_metro, 'sr', markerfacecolor='none', markeredgecolor='
    g', markeredgewidth='2', label = r'$E_{T}$' + ' / ' + r'$P_{Exp}$' + ' '
    + r'$(\chi^{2}=%5.3e)$' % tuple([chi2_teo]), markersize=20)
# Ajuste - curve_fit
ax1.plot(es_cf, hist_metro, 'bo', markeredgecolor='k', markeredgewidth='1',
    label = r'$E_{fit}$' + ' / ' + r'$P_{Exp}$' + ' (cf: J=%5.3f, ' % tuple(
    J_cf) + r'$\chi^{2}_{std}=%5.3e)$' % tuple([chi2_cf]), markersize=12)
# Línea teórica
ax1.plot(es_teo, ps_teo, 'r-', label = r'$E_{T}$' + ' / ' + r'$P_{T}$',
    linewidth=8, alpha=0.5)
ax1.legend(loc='upper right', fontsize=15)

# Almacenamiento
graphXa_imgname = 'tesis_IsingTest_scEG_chi2_std_L%(length)i_J%(coupling).3
    f_Jguess%(guess).3f_Nrondas%(turns).3e_seed%(seed)i_c%(cte).3
    f_LS_CurveFit_GraphTotalX.png'
```

```
609 plt.savefig(graphXa_imgname % file_dict)
610
611 ### GRÁFICA CON AJUSTE (Minimize Scalar)
612 plt.figure(2)
613 #fig3, ((ax3a, ax3b), (ax3c, ax3d), (ax3e, ax3f), (ax3g, ax3h)) = plt.
    subplots(nrows=4, ncols=2, figsize=(30,60))
614 fig2, ax2 = plt.subplots(nrows=1, ncols=1, figsize=(10,10))
615 #fig3, ax3a = plt.subplots(nrows=1, ncols=1, figsize=(18,18))
616 plt.rcParams.update({'font.size': 24})
617 plt.rcParams.update({'xtick.labelsize': 22})
618 plt.rcParams.update({'ytick.labelsize': 22})
619 plt.tight_layout(pad=4, h_pad=4, w_pad=4)
620 ###################
621 #### Inclusivo ####
622 ###################
623 ax2.set_title('Ising ' + r'$(J=%5.3f, J_{g}=%5.3f)$' % tuple([J,Jguess]))
624 ax2.set_yscale('log')
625 ax2.set_ylabel(r'$P(S_{i})$', fontsize=22)
626 max_prob = max(hist_metro)
627 min_prob = minPos(hist_metro)
628 ax2.set_ylim(bottom=min_prob/10.0, top=max_prob*10)
629 ax2.set_xlabel(r'$H(S_{i})$',fontsize=22)
630 minE_inc = min(min(fitE_values_cf), min(fitE_values_ms), min(fitE_values_min
    ), min(E_values))
631 maxE_inc = max(max(fitE_values_cf), max(fitE_values_ms), min(fitE_values_min
    ), max(E_values))
632 delE_inc = (maxE_inc - minE_inc) / 10.0
633 ax2.set_xlim(minE_inc - delE_inc, maxE_inc + delE_inc)
634 ax2.tick_params(axis='both', which='major', labelsize=20)
635 # Teóricos
636 ax2.plot(es_teo, hist_metro, 'sr', markerfacecolor='none', markeredgecolor='
    g', markeredgewidth='2', label = r'$E_{T}$' + ' / ' + r'$P_{Exp}$' + ' '
    + r'$(\chi^{2}=%5.3e)$' % tuple([chi2_teo]), markersize=20)
637 # Ajuste - minimize_scalar
638 ax2.plot(es_ms, hist_metro, 'bo', markeredgecolor='k', markeredgewidth='1',
    label = r'$E_{Fit}$' + ' / ' + r'$P_{Exp}$' + ' (ms: J=%5.3f, ' % tuple(
    J_ms) + r'$\chi^{2}_{std}=%5.3e)$' % tuple([chi2_ms]), markersize=12)
639 # Línea teórica
640 ax2.plot(es_teo, ps_teo, 'r-', label = r'$E_{T}$' + ' / ' + r'$P_{T}$',
    linewidth=8, alpha=0.5)
641 ax2.legend(loc='upper right', fontsize=15)
642
643 # Almacenamiento
644 graphXb_imgname = 'tesis_IsingTest_scEG_chi2_std_L%(length)i_J%(coupling).3
    f_Jguess%(guess).3f_Nrondas%(turns).3e_seed%(seed)i_c%(cte).3
    f_LS_MinimizeScalar_GraphTotalX.png'
645 plt.savefig(graphXb_imgname % file_dict)
646
647
648 ### GRÁFICA CON AJUSTE (Minimize)
649 plt.figure(3)
650 #fig3, ((ax3a, ax3b), (ax3c, ax3d), (ax3e, ax3f), (ax3g, ax3h)) = plt.
    subplots(nrows=4, ncols=2, figsize=(30,60))
651 fig3, ax3 = plt.subplots(nrows=1, ncols=1, figsize=(10,10))
652 #fig3, ax3a = plt.subplots(nrows=1, ncols=1, figsize=(18,18))
653 plt.rcParams.update({'font.size': 24})
654 plt.rcParams.update({'xtick.labelsize': 22})
655 plt.rcParams.update({'ytick.labelsize': 22})
```

```
656 plt.tight_layout(pad=4, h_pad=4, w_pad=4)
657 ###################
658 #### Inclusivo ####
659 ###################
660 ax3.set_title('Ising ' + r'$(J=%5.3f, J_{g}=%5.3f)$' % tuple([J,Jguess]))
661 ax3.set_yscale('log')
662 ax3.set_ylabel(r'$P(S_{i})$', fontsize=22)
663 max_prob = max(hist_metro)
664 min_prob = minPos(hist_metro)
665 ax3.set_ylim(bottom=min_prob/10.0, top=max_prob*10)
666 ax3.set_xlabel(r'$H(S_{i})$',fontsize=22)
667 minE_inc = min(min(fitE_values_cf), min(fitE_values_ms), min(fitE_values_min
    ), min(E_values))
668 maxE_inc = max(max(fitE_values_cf), max(fitE_values_ms), min(fitE_values_min
    ), max(E_values))
669 delE_inc = (maxE_inc - minE_inc) / 10.0
670 ax3.set_xlim(minE_inc - delE_inc, maxE_inc + delE_inc)
671 ax3.tick_params(axis='both', which='major', labelsize=20)
672 # Teóricos
673 ax3.plot(es_teo, hist_metro, 'sr', markerfacecolor='none', markeredgecolor='
    g', markeredgewidth='2', label = r'$E_{T}$' + ' / ' + r'$P_{Exp}$' + ' '
    + r'$(\chi^{2}=%5.3e)$' % tuple([chi2_teo]), markersize=20)
674 # Ajuste - minimize
675 ax3.plot(es_min, hist_metro, 'bo', markeredgecolor='k', markeredgewidth='1',
     label = r'$E_{Fit}$' + ' / ' + r'$P_{Exp}$' + ' (m: J=%5.3f, ' % tuple(
    J_min) + r'$\chi^{2}_{std}=%5.3e)$' % tuple([chi2_min]), markersize=12)
676 # Línea teórica
677 ax3.plot(es_teo, ps_teo, 'r-', label = r'$E_{T}$' + ' / ' + r'$P_{T}$',
    linewidth=8, alpha=0.5)
678 ax3.legend(loc='upper right', fontsize=15)
679
680 # Almacenamiento
681 graphXc_imgname = 'tesis_IsingTest_scEG_chi2_std_L%(length)i_J%(coupling).3
    f_Jguess%(guess).3f_Nrondas%(turns).3e_seed%(seed)i_c%(cte).3
    f_LS_Minimize_GraphTotalX.png'
682 plt.savefig(graphXc_imgname % file_dict)
```

Apéndice L

Gráficas de simulaciones para prueba de distribución de Boltzmann en rejillas de mapeo acoplado "a grano grueso" bajo los cuatro esquemas de minimización

Las gráficas en las siguientes figuras han sido etiquetadas de acuerdo al esquema empleado para determinar los valores óptimos de los coeficientes $\alpha(d)$:

- (a) Esquema local 0: El método que regulaba las variaciones en los coeficientes $\alpha(d)$ efectuadas por `minimize` no fue suministrado y todos los coeficientes α podían ser equivalentes a cualquier número real.

- (b) Esquema local 1: La función `minimize` operó con el método `trust-constr` sobre todos los coeficientes α simultáneamente.

- (c) Esquema gradual 0: Sin haber proporcionado un método específico a `minimize`, los coeficientes $\alpha(d)$ fueron determinados de uno en uno.

- (d) Esquema gradual 1: `minimize` definió un coeficiente a la vez implementando el método `trust-constr`.

L.1. Resultados de sistema CG-CML construido a partir de una rejilla de mapeo acoplado de longitud $L = 8$, grano $G = 2$ y acoplamiento $g = 0.204$

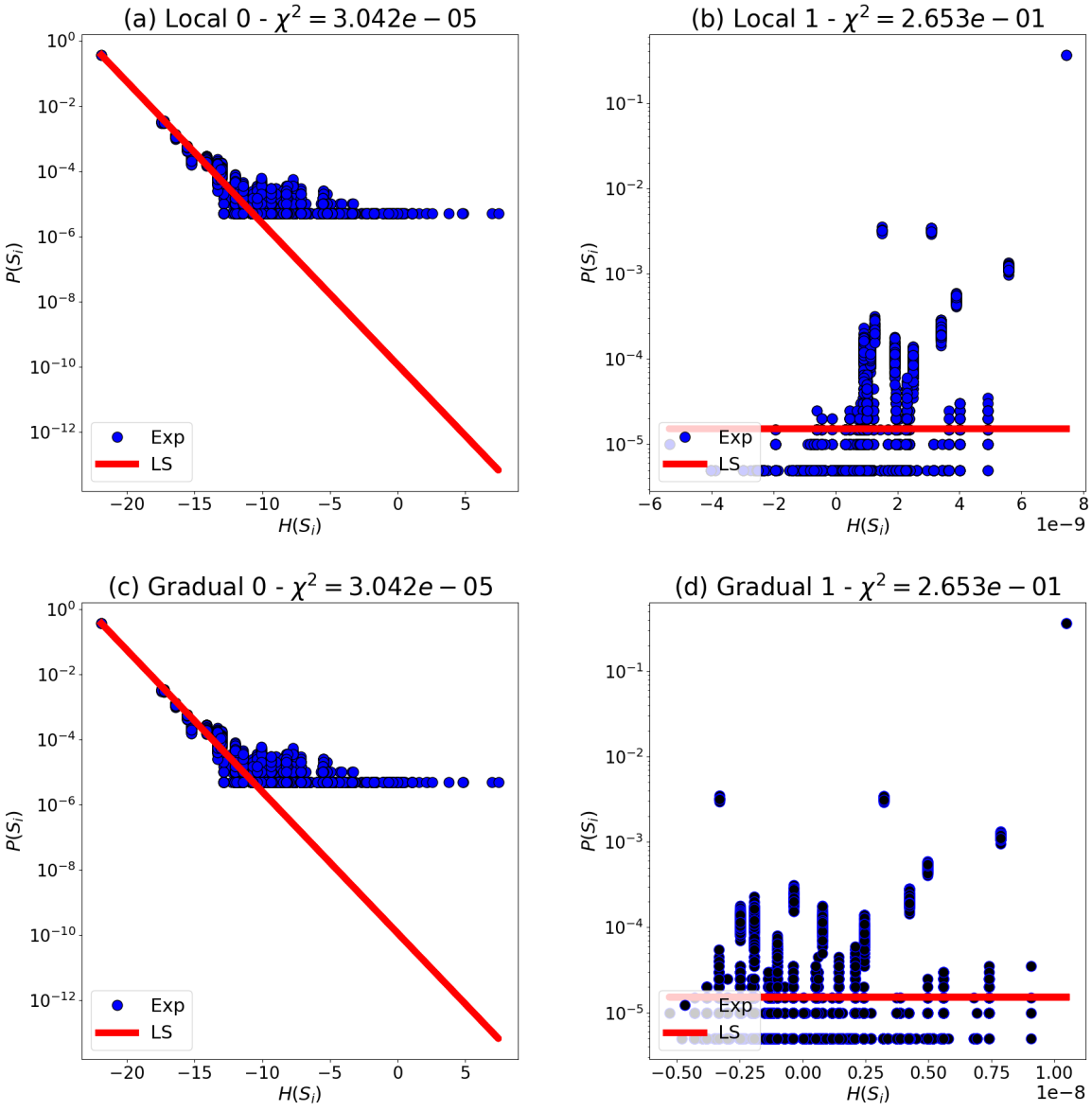


Figura L.1: $\left\langle \left(\Delta^{(i,j)}/N^{(i,j)}\right)^2 \right\rangle$ en sistema CG-CML de 4×4 sitios

Figura L.2: Relación entre las probabilidades $\mathcal{P}(\mathcal{S}_i)$ de los estados S_i y las energías $\mathcal{H}(S_i)$ disponibles a un sistema CG-CML de 4×4 sitios con una evolución temporal de 2×10^5 iteraciones y semilla $s = 3249191637585704358$. Tal sistema fue generado a partir de una rejilla de mapeo acoplado con longitud $L = 8$, grano $G = 2$ y acoplamiento $g = 0.204$. Los puntos azules corresponden a las mediciones de las probabilidades de cada energía tras el ajuste y la recta roja marca la distribución de Boltzmann asociada a los coeficientes $\alpha(d)$ ajustados.

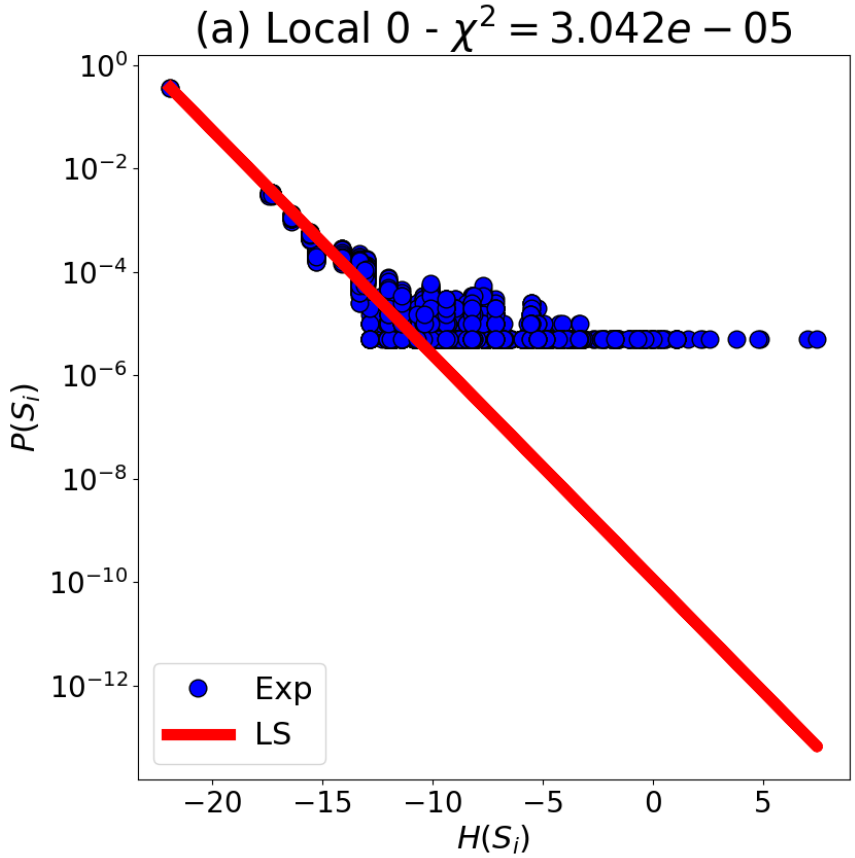


Figura L.3: Distribución de probabilidades $\mathcal{P}(\mathcal{S}_i)$ de los estados S_i en términos de las energías $\mathcal{H}(S_i)$ disponibles a un sistema CG-CML de 4×4 sitios con una evolución temporal de 2×10^5 iteraciones y semilla $s = 3249191637585704358$, usando el esquema de minimización local 0. Tal sistema fue generado a partir de una rejilla de mapeo acoplado con longitud $L = 8$, grano $G = 2$ y acoplamiento $g = 0.204$. Los puntos azules corresponden a las mediciones de las probabilidades de cada energía tras el ajuste y la recta roja marca la distribución de Boltzmann asociada a los coeficientes $\alpha(d)$ ajustados.

Método	$\alpha(1)$	$\alpha(\sqrt{2})$	$\alpha(2)$	$\alpha(\sqrt{5})$	$\alpha(\sqrt{8})$	χ^2
Local 0	-1.1507E+00	3.9358E-01	1.4258E-01	6.3760E-02	-2.4802E-01	3.0417E-05
Gradual 0	-1.1507E+00	3.9357E-01	1.4257E-01	6.3777E-02	-2.4805E-01	3.0416E-05
Local 1	4.2851E-11	7.2450E-11	3.0028E-10	-1.5109E-11	-7.0469E-11	2.6535E-01
Gradual 1	-6.8332E-11	9.1560E-11	5.4079E-10	-7.1040E-11	4.1901E-10	2.6535E-01

Cuadro L.1: Acoplamientos $\alpha(d)$ ajustados a distribución experimental de probabilidades $\mathcal{P}(S_i)$ de sistema CG-CML de 4×4 sitios con una evolución temporal de 2×10^5 iteraciones, elaborado a partir de una rejilla de mapeo acoplado de longitud $L = 8$, tamaño de grano $G = 2$ y acoplamiento $g = 0.204$

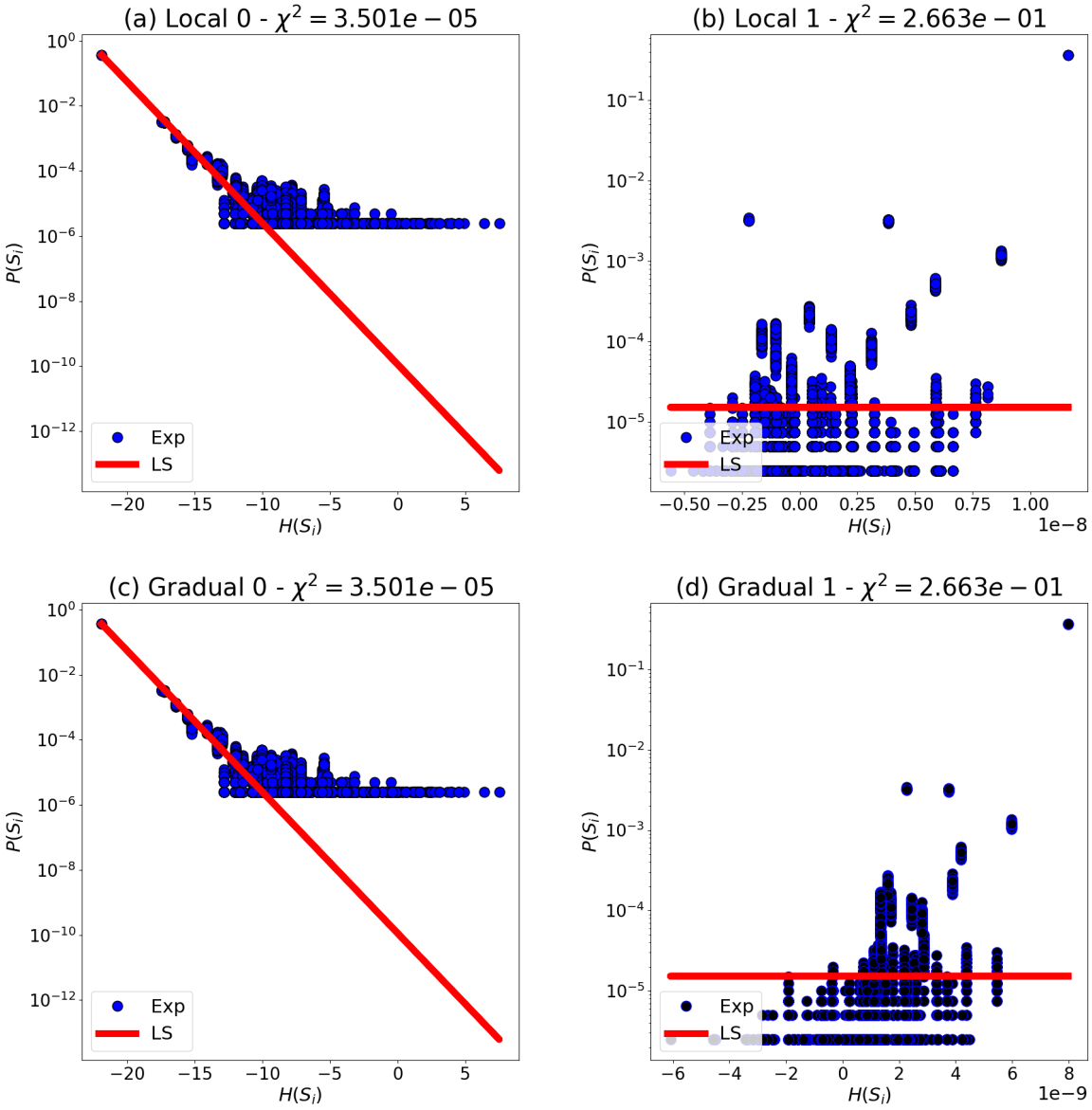


Figura L.4: Relación entre las probabilidades $\mathcal{P}(\mathcal{S}_i)$ de los estados S_i y las energías $\mathcal{H}(S_i)$ disponibles a un sistema CG-CML de 4×4 sitios con una evolución temporal de 4×10^5 iteraciones y semilla $s = 5617058624460882590$. Tal sistema fue generado a partir de una rejilla de mapeo acoplado con longitud $L = 8$, grano $G = 2$ y acoplamiento $g = 0.204$. Los puntos azules corresponden a las mediciones de las probabilidades de cada energía tras el ajuste y la recta roja marca la distribución de Boltzmann asociada a los coeficientes $\alpha(d)$ ajustados.

L.2. Resultados de sistema CG-CML construido a partir de una rejilla de mapeo acoplado de longitud $L = 32$, grano $G = 8$ y acoplamiento $g = 0.204$

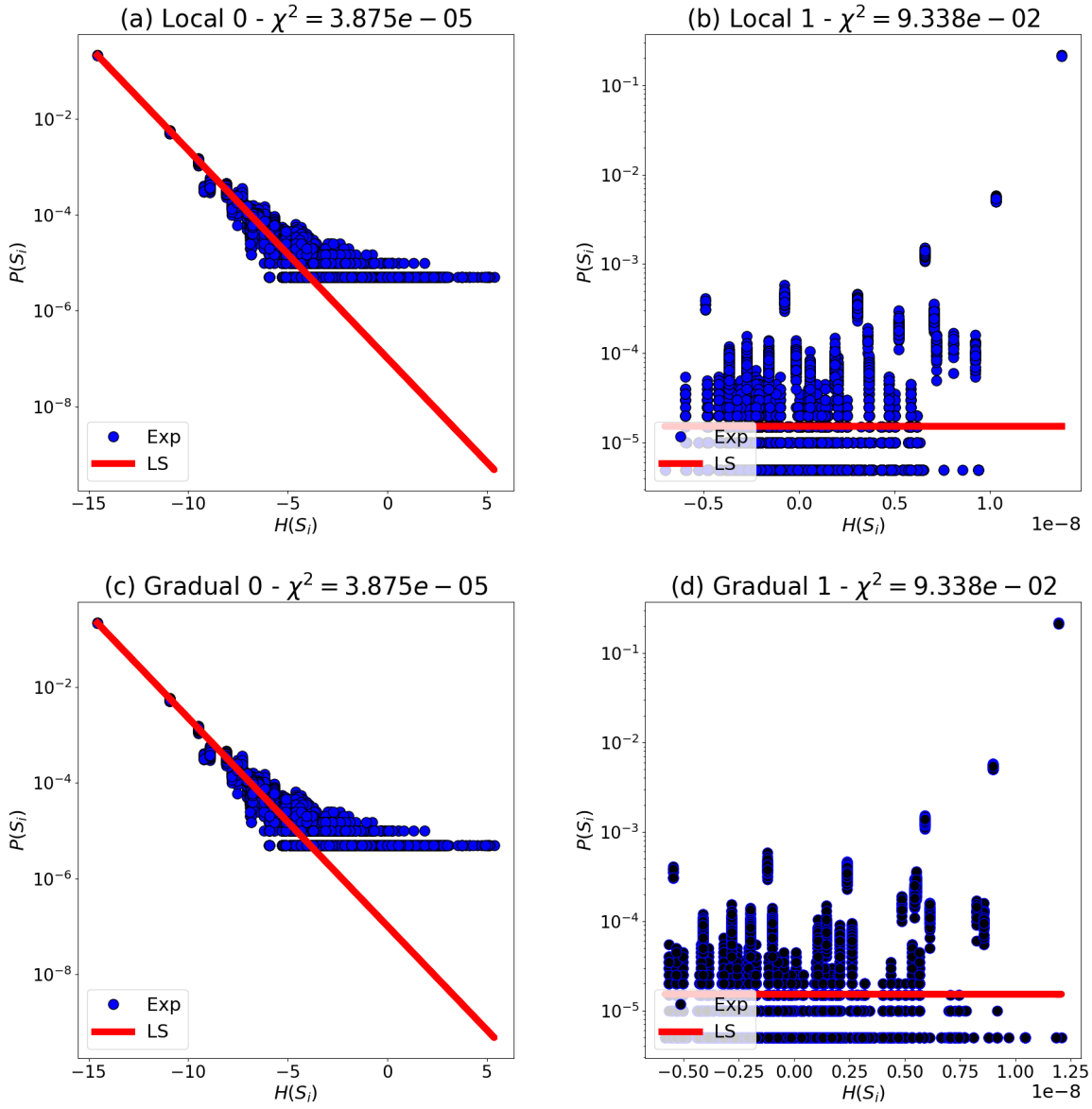


Figura L.5: Relación entre las probabilidades $\mathcal{P}(\mathcal{S}_i)$ de los estados S_i y las energías $\mathcal{H}(S_i)$ disponibles a un sistema CG-CML de 4×4 sitios con una evolución temporal de 2×10^5 iteraciones y semilla $s = 3316581566959966622$. Tal sistema fue generado a partir de una rejilla de mapeo acoplado con longitud $L = 32$, grano $G = 8$ y acoplamiento $g = 0.204$. Los puntos azules corresponden a las mediciones de las probabilidades de cada energía tras el ajuste y la recta roja marca la distribución de Boltzmann asociada a los coeficientes $\alpha(d)$ ajustados.

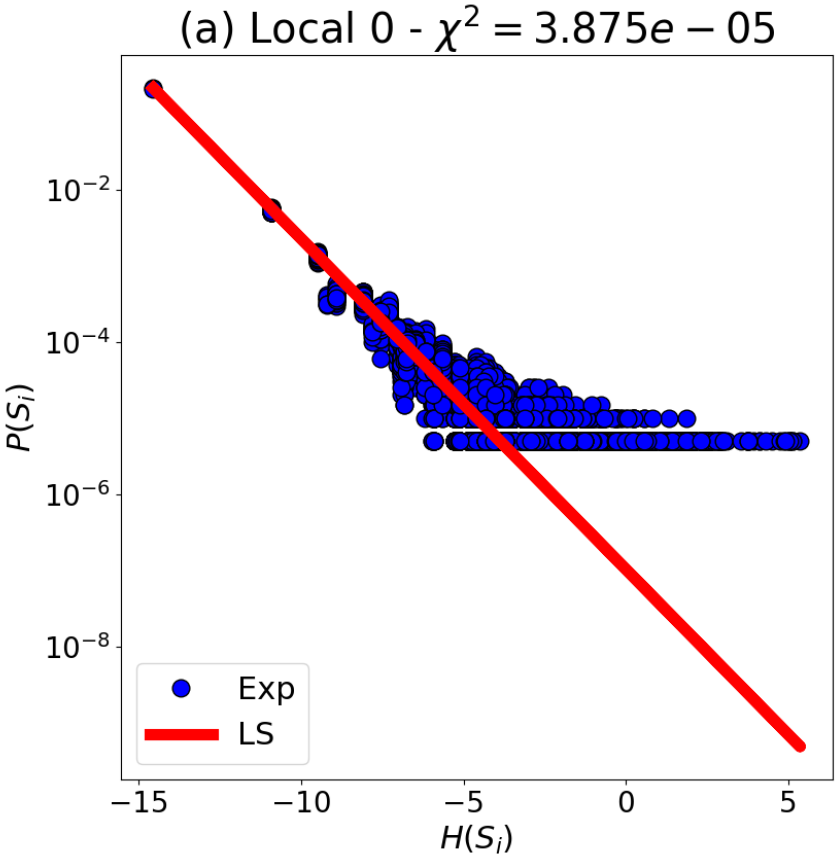


Figura L.6: Distribución de probabilidades $\mathcal{P}(\mathcal{S}_i)$ de los estados S_i en términos de las energías $\mathcal{H}(S_i)$ disponibles a un sistema CG-CML de 4×4 sitios con una evolución temporal de 2×10^5 iteraciones y semilla $s = 3316581566959966622$, usando el esquema de minimización local 0. Tal sistema fue generado a partir de una rejilla de mapeo acoplado con longitud $L = 32$, grano $G = 8$ y acoplamiento $g = 0.204$. Los puntos azules corresponden a las mediciones de las probabilidades de cada energía tras el ajuste y la recta roja marca la distribución de Boltzmann asociada a los coeficientes $\alpha(d)$ ajustados.

Método	$\alpha(1)$	$\alpha(\sqrt{2})$	$\alpha(2)$	$\alpha(\sqrt{5})$	$\alpha(\sqrt{8})$	χ^2
Local 0	-5.5594E-01	-3.3749E-03	1.3565E-01	5.3624E-02	-6.7698E-02	3.8753E-05
Gradual 0	-5.5593E-01	-3.3855E-03	1.3562E-01	5.3655E-02	-6.7743E-02	3.8753E-05
Local 1	-7.1539E-11	4.6800E-11	5.9189E-10	8.1959E-11	3.0542E-10	9.3382E-02
Gradual 1	-1.9384E-11	-1.1168E-10	6.4835E-10	3.8680E-11	5.6796E-10	9.3382E-02

Cuadro L.2: Acoplamientos $\alpha(d)$ ajustados a distribución experimental de probabilidades $\mathcal{P}(S_i)$ de sistema CG-CML de 4×4 sitios con una evolución temporal de 2×10^5 iteraciones, elaborado a partir de una rejilla de mapeo acoplado de longitud $L = 32$, tamaño de grano $G = 8$ y acoplamiento $g = 0.204$

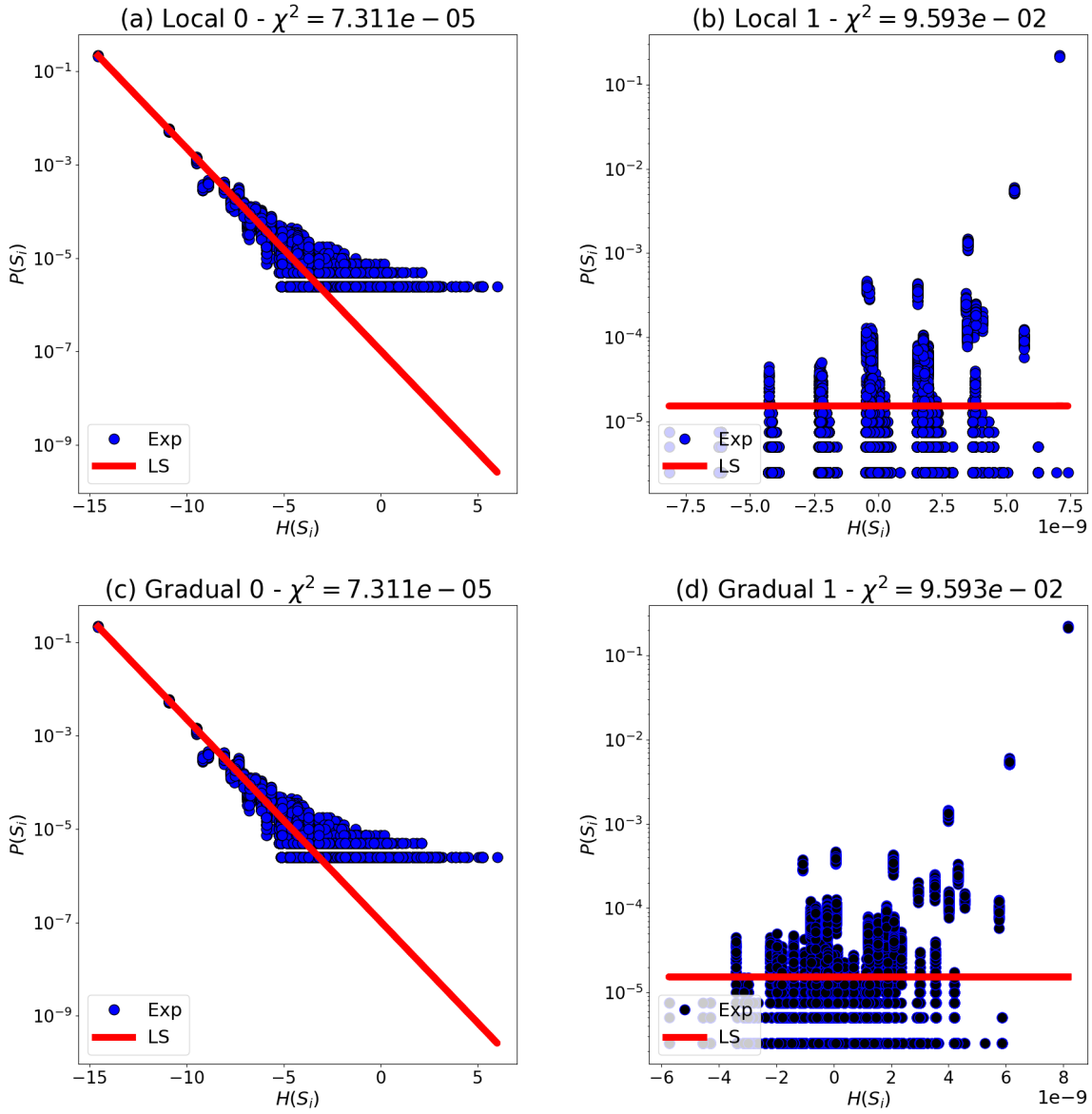


Figura L.7: Relación entre las probabilidades $\mathcal{P}(\mathcal{S}_i)$ de los estados S_i y las energías $\mathcal{H}(S_i)$ disponibles a un sistema CG-CML de 4×4 sitios con una evolución temporal de 4×10^5 iteraciones y semilla $s = 3759620218372826529$. Tal sistema fue generado a partir de una rejilla de mapeo acoplado con longitud $L = 32$, grano $G = 8$ y acoplamiento $g = 0.204$. Los puntos azules corresponden a las mediciones de las probabilidades de cada energía tras el ajuste y la recta roja marca la distribución de Boltzmann asociada a los coeficientes $\alpha(d)$ ajustados.

L.3. Resultados de sistema CG-CML construido a partir de una rejilla de mapeo acoplado de longitud $L = 32$, grano $G = 8$ y acoplamiento $g = 0$

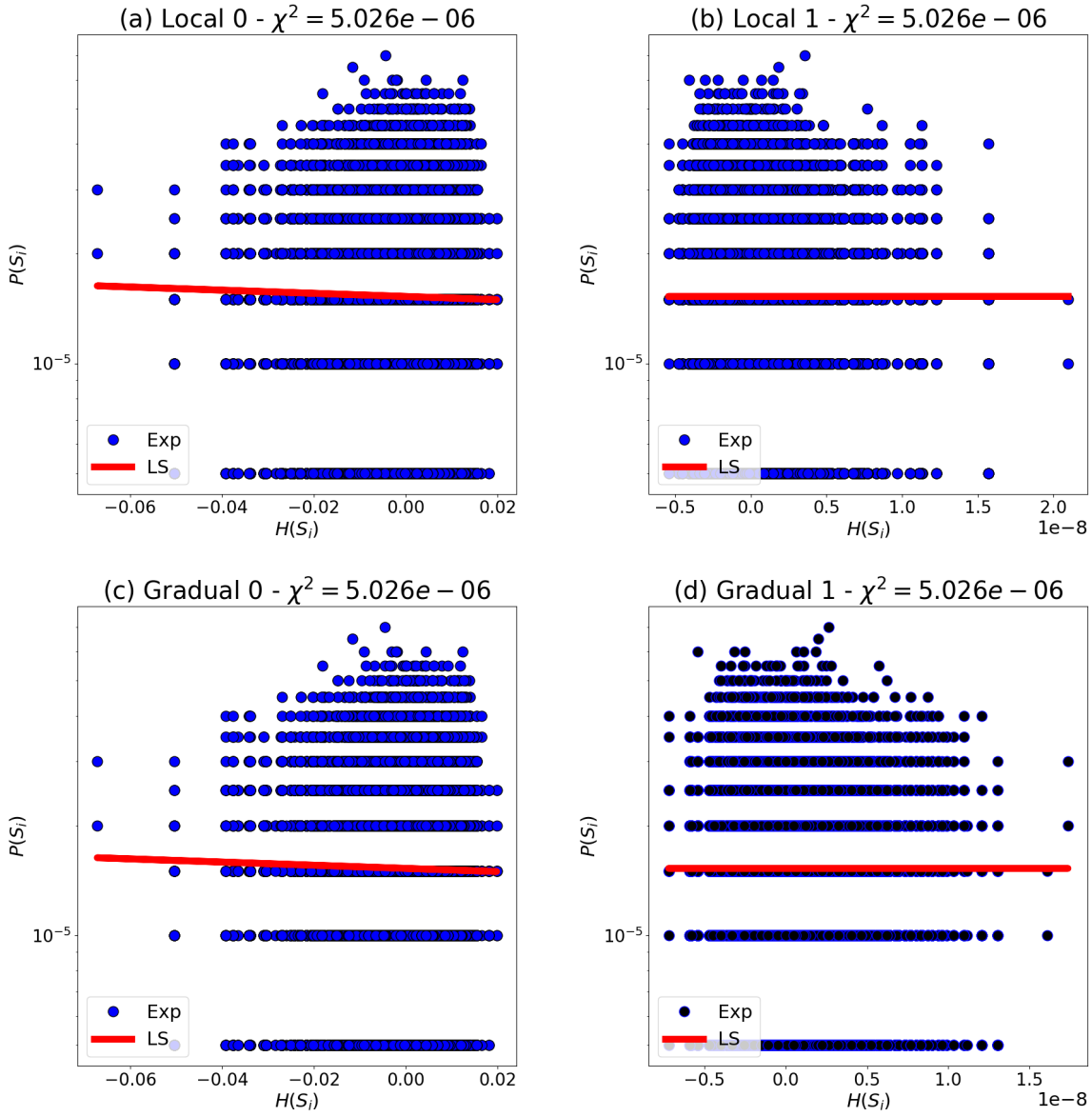


Figura L.8: Relación entre las probabilidades $\mathcal{P}(\mathcal{S}_i)$ de los estados S_i y las energías $\mathcal{H}(S_i)$ disponibles a un sistema CG-CML de 4×4 sitios con una evolución temporal de 2×10^5 iteraciones y semilla $s = 6197905548778816112$. Tal sistema fue generado a partir de una rejilla de mapeo acoplado con longitud $L = 32$, grano $G = 8$ y acoplamiento $g = 0$. Los puntos azules corresponden a las mediciones de las probabilidades de cada energía tras el ajuste y la recta roja marca la distribución de Boltzmann asociada a los coeficientes $\alpha(d)$ ajustados.

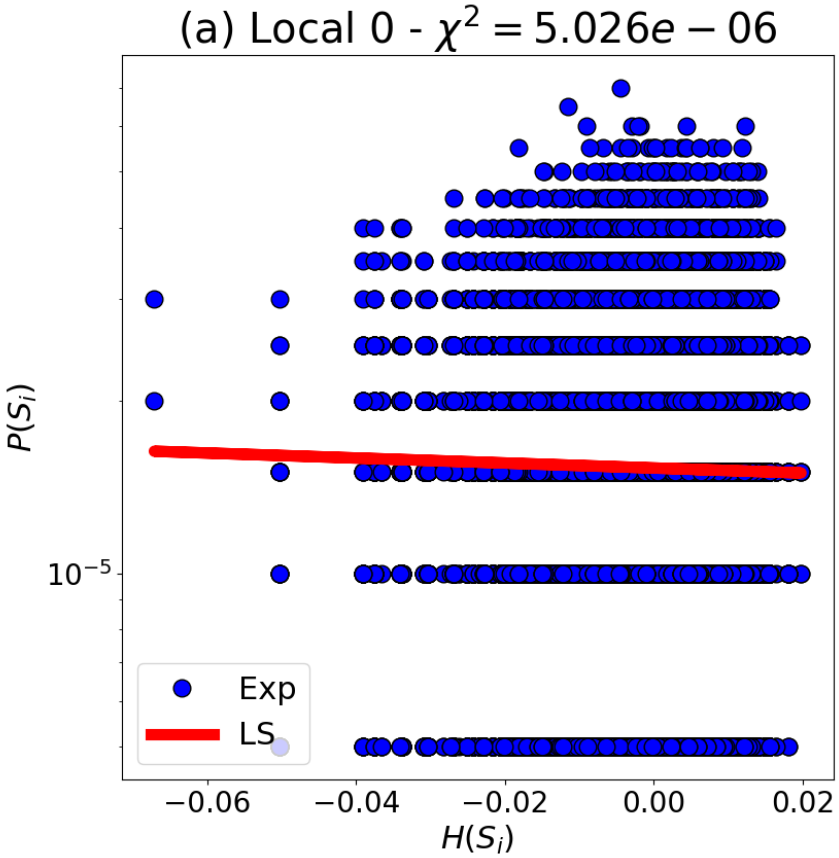


Figura L.9: $\left\langle \left(\Delta^{(i,j)}/N^{(i,j)}\right)^2 \right\rangle$ en sistema CG-CML de 4×4 sitios

Figura L.10: Distribución de probabilidades $\mathcal{P}(\mathcal{S}_i)$ de los estados S_i en términos de las energías $\mathcal{H}(S_i)$ disponibles a un sistema CG-CML de 4×4 sitios con una evolución temporal de 2×10^5 iteraciones y semilla $s = 6197905548778816112$, usando el esquema de minimización local 0. Tal sistema fue generado a partir de una rejilla de mapeo acoplado con longitud $L = 32$, grano $G = 8$ y acoplamiento $g = 0$. Los puntos azules corresponden a las mediciones de las probabilidades de cada energía tras el ajuste y la recta roja marca la distribución de Boltzmann asociada a los coeficientes $\alpha(d)$ ajustados.

Método	$\alpha(1)$	$\alpha(\sqrt{2})$	$\alpha(2)$	$\alpha(\sqrt{5})$	$\alpha(\sqrt{8})$	χ^2
Local 0	1.4067E-04	-6.6447E-05	-1.3978E-03	1.0054E-03	-7.4231E-04	5.0256E-06
Gradual 0	1.3689E-04	-7.3613E-05	-1.4032E-03	1.0040E-03	-7.4070E-04	5.0256E-06
Local 1	1.1626E-11	2.0604E-10	4.4641E-10	1.6032E-10	2.1396E-10	5.0258E-06
Gradual 1	-8.5312E-11	1.7386E-10	5.7332E-10	6.5242E-11	2.4689E-10	5.0258E-06

Cuadro L.3: Acoplamientos $\alpha(d)$ ajustados a distribución experimental de probabilidades $\mathcal{P}(S_i)$ de sistema CG-CML de 4×4 sitios con una evolución temporal de 2×10^5 iteraciones, elaborado a partir de una rejilla de mapeo acoplado de longitud $L = 32$, tamaño de grano $G = 8$ y acoplamiento $g = 0.000$

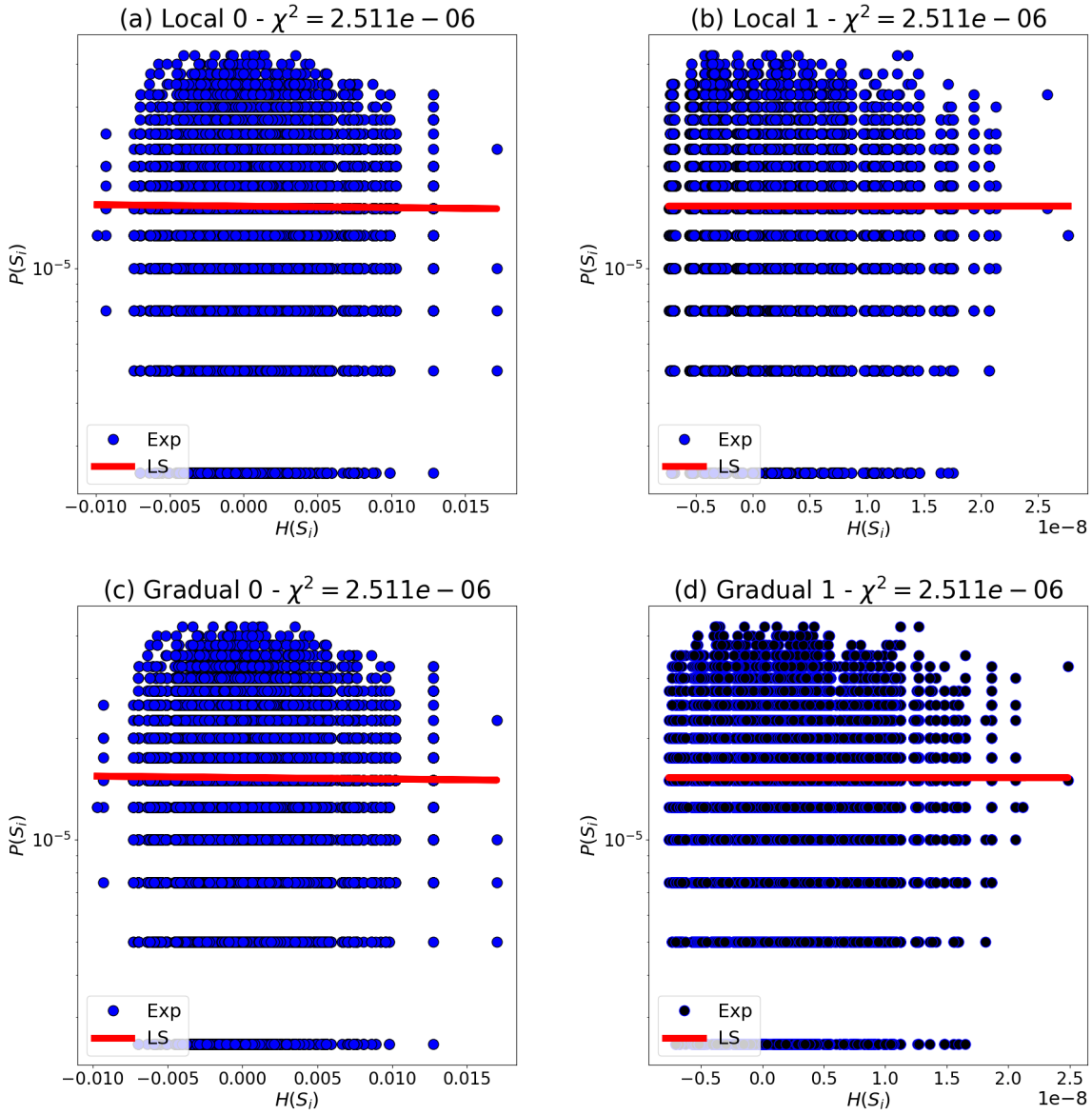


Figura L.11: Relación entre las probabilidades $\mathcal{P}(\mathcal{S}_i)$ de los estados S_i y las energías $\mathcal{H}(S_i)$ disponibles a un sistema CG-CML de 4×4 sitios con una evolución temporal de 4×10^5 iteraciones y semilla $s = 4523530975986176005$. Tal sistema fue generado a partir de una rejilla de mapeo acoplado con longitud $L = 32$, grano $G = 8$ y acoplamiento $g = 0$. Los puntos azules corresponden a las mediciones de las probabilidades de cada energía tras el ajuste y la recta roja marca la distribución de Boltzmann asociada a los coeficientes $\alpha(d)$ ajustados.

Apéndice M

Código para prueba de distribución de Boltzmann para los estados de rejilla de mapeo acoplado “a grano grueso” considerando distintos números de acoplamientos entre sitios

```
import sys
import numpy as np
import networkx as nx
import random
import matplotlib.pyplot as plt
import scipy
from scipy.optimize import minimize
from scipy.optimize import Bounds
from scipy.optimize import LinearConstraint
import json

##############################################################################
##############################################################################

# Cálculo de distancias entre elementos en una rejilla cuadrada
    bidimensional bajo condiciones de frontera periódicas
def calc_dist(loc1, loc2, longitud):
  distx = loc1[0] - loc2[0]
  disty = loc1[1] - loc2[1]
  dist1 = (distx ** 2 + disty ** 2) ** (1 / 2.0)
  dist2 = ((distx - longitud) ** 2 + disty ** 2) ** (1 / 2.0)
  dist3 = ((distx + longitud) ** 2 + disty ** 2) ** (1 / 2.0)
  dist4 = (distx ** 2 + (disty - longitud) ** 2) ** (1 / 2.0)
  dist5 = (distx ** 2 + (disty + longitud) ** 2) ** (1 / 2.0)
  dist6 = ((distx + longitud) ** 2 + (disty + longitud) ** 2) ** (1 / 2.0)
  dist7 = ((distx - longitud) ** 2 + (disty + longitud) ** 2) ** (1 / 2.0)
  dist8 = ((distx - longitud) ** 2 + (disty - longitud) ** 2) ** (1 / 2.0)
  dist9 = ((distx + longitud) ** 2 + (disty - longitud) ** 2) ** (1 / 2.0)
  lista_distancias = [dist1, dist2, dist3, dist4, dist5, dist6, dist7, dist8
    , dist9]
  distancia = min(lista_distancias)
  return distancia
```

```
31
32 # Iniciación de gráfica para rejilla CML a grano grueso en estado 0
33 def ini_graf(longitud):
34   grafica = nx.Graph()
35   num_sitios = longitud**2
36   # Definición de nodos
37   for sitio in range(num_sitios):
38     grafica.add_node(sitio, u = -1)
39   # Definición de edges
40   lista_edges = []
41   for sitio1 in range(num_sitios):
42     for sitio2 in range(num_sitios):
43       if sitio2 == sitio1:
44         continue
45       coord1 = (sitio1 % longitud, sitio1 // longitud)
46       coord2 = (sitio2 % longitud, sitio2 // longitud)
47       distancia = calc_dist(coord1, coord2, longitud)
48       lista_edges.append((sitio1, sitio2, {'path_distance': distancia}))
49   grafica.add_edges_from(lista_edges)
50   return grafica
51
52 # Configuración de gráfica en un estado definido
53 def def_edo(estado, grafica):
54   referencia = estado
55   exp = 0
56   while referencia > 0:
57     factor2 = 2 ** (exp)
58     tasa = referencia / factor2
59     if tasa >= 1:
60       exp = exp + 1
61     else:
62       grafica.nodes[exp - 1]['u'] = 1
63       factorneg = 2 ** (exp - 1)
64       referencia = referencia - factorneg
65       exp = 0
66   return grafica
67
68 # Definición de lista con distancias posibles entre sitios de la gráfica
69 def list_dists(grafica):
70   lista_distancias = []
71   for (nodo1, nodo2, distancia) in grafica.edges.data('path_distance'):
72     if distancia not in lista_distancias:
73       lista_distancias.append(distancia)
74     else:
75       continue
76   lista_distancias = sorted(lista_distancias)
77   return lista_distancias
78
79 ### PROBABILIDADES DE BOLTZMANN PARA ENERGÍAS
80 def prob_boltzmann(arreglo_alfas):
81   # Definición de diccionario con coeficientes alfas
82   diccionario_alfas = {}
83   for dist_iter in range(len(list_key_dists)):
84     clave_dist = list_key_dists[dist_iter]
85     diccionario_alfas[clave_dist] = arreglo_alfas[dist_iter]
86   # Cálculo de probabilidades de Boltzmann para todos los estados diferentes
     que están disponibles al sistema
87   p_calc = np.zeros(len(list_sums))
```

```
88    funcion_Z = 0
89    contador_p = 0
90    for entry in protoE:
91      # Cálculo de Hamiltoniano para un estado particular dadas ciertas alfas
92      hamiltoniano = 0
93      for distancia in list_key_dists:
94        factor = diccionario_alfas[distancia] * entry[0][distancia]
95        hamiltoniano = hamiltoniano + factor
96      # Definición (parcial) de probabilidad para cada estado
97      p_calc[contador_p] = entry[2] * np.exp(-hamiltoniano)
98      # Definición (gradual) de función de partición Z
99      funcion_Z = funcion_Z + (entry[2] * np.exp(-hamiltoniano))
100     contador_p = contador_p + 1
101   # Definición de probabilidad de Boltzmann para todos los estados
      disponibles al sistema
102   p_calc = p_calc / funcion_Z
103   return p_calc, funcion_Z
104
105 # Mínimos cuadrados
106 def min_cuadrados(arreglo_alfas):
107   prob_calc, f_Z = prob_boltzmann(arreglo_alfas)
108   dif = 0
109   for i in range(len(prob_calc)):
110 #     if protoE[i][1] == 0:
111 #       continue
112     dif = dif + ((prob_calc[i] - protoE[i][1])) ** 2
113   return dif
114
115 ### PROBABILIDADES DE BOLTZMANN PARA ESTADOS
116 def pe_estado(arreglo_alfas):
117   # Definición de diccionario con coeficientes alfas
118   diccionario_alfas = {}
119   for dist_iter in range(len(list_key_dists)):
120     clave_dist = list_key_dists[dist_iter]
121     diccionario_alfas[clave_dist] = arreglo_alfas[dist_iter]
122   # Cálculo de probabilidades de Boltzmann para todos los estados diferentes
      que están disponibles al sistema
123   p_edo = np.zeros(N_total_states)
124   e_edo = np.zeros(N_total_states)
125   funcion_Z = 0
126   contador_p = 0
127   for estado in range(N_total_states):
128     clave = 's' + str(float(estado))
129     # Cálculo de Hamiltoniano para un estado particular dadas ciertas alfas
130     hamiltoniano = 0
131     for distancia in list_key_dists:
132       factor = diccionario_alfas[distancia] * dict_sums[clave][distancia]
133       hamiltoniano = hamiltoniano + factor
134     # Definición (parcial) de probabilidad para cada estado
135     p_edo[contador_p] = np.exp(-hamiltoniano)
136     # Definición de energía para cada estado
137     e_edo[contador_p] = hamiltoniano
138     # Definición (gradual) de función de partición Z
139     funcion_Z = funcion_Z + np.exp(-hamiltoniano)
140     contador_p = contador_p + 1
141   # Definición de probabilidad de Boltzmann para todos los estados
      disponibles al sistema
142   p_edo = p_edo / funcion_Z
```

```
143   return e_edo, p_edo, funcion_Z
144
145 ### FUNCIÓN PARA ORDENAR ARREGLOS
146 def ordenador(energias_ajuste, probabilidades_ajuste,
      probabilidades_experimentales):
147   # Ordenamiento de energías resultantes de ajuste
148   energias_ordenadas = sorted(energias_ajuste)
149   # Identificación de índices
150   indices = []
151   for energia_ord in energias_ordenadas:
152     for ind, energia in enumerate(energias_ajuste):
153       if energia == energia_ord:
154         indices.append(ind)
155         break
156   # Ordenamiento de probabilidades
157   prob_ajuste_ordenadas = []
158   prob_exp_ordenadas = []
159   for ind in indices:
160     prob_ajuste_ordenadas.append(probabilidades_ajuste[ind])
161     prob_exp_ordenadas.append(probabilidades_experimentales[ind])
162   return energias_ordenadas, prob_ajuste_ordenadas, prob_exp_ordenadas
163
164 ##############################################################################
165 ##############################################################################
166
167 ### RECUPERACIÓN DE EVOLUCIÓN DE REJILLA CML DE GRANO GRUESO
168 # Solicitud de parámetros de rejilla CML
169 L = int(input('Ingresa la longitud de la rejilla (entero): '))
170 G = int(input('Ingresa el tamaño de grano (entero): '))
171 N_total_sites = int(L ** 2 / G ** 2)
172 N_total_states = 2 ** (N_total_sites)
173 g = float(input('Ingresa la constante de acoplamiento (real con tres
      decimales): '))
174 N_iter = int(input('Ingresa el total de iteraciones (entero): '))
175 transient = int(input('Ingresa el valor de transient (entero): '))
176 N_turns = int(input('Ingresa el total de rondas (entero): '))
177 n_tries = int(input('Ingresa el número de ansatz (entero): '))
178 seed = int(input('Ingresa el valor de la semilla (entero): '))
179 sref = random.randrange(sys.maxsize)
180 random.seed(sref)
181 # Lectura de archivo con datos
182 fname_listofstates_CG = './Res_BD_L%(length)i_G%(grainsize)i_g%(coupling).3
      f_Niter%(iterations).3e_trans%(trans).3e_Nrondas%(turns).3e_seed%(s)i/
      tesis_Egolf_baldet_L%(length)i_G%(grainsize)i_g%(coupling).3f_Niter%(
      iterations).3e_trans%(trans).3e_Nrondas%(turns).3e_seed%(s)
      i_ListOfStates_CG.txt'
183 dict_read = {'length': L, 'grainsize': G, 'coupling': g, 'iterations':
      N_iter, 'trans': transient, 'turns': N_turns, 's': seed}
184 # Definición de arreglo con evolución de rejilla CML de grano grueso
185 arr_CG_states = np.loadtxt(fname_listofstates_CG % dict_read)
186
187 ### DISTRIBUCIÓN DE PROBABILIDAD DADA LA EVOLUCIÓN DE LA REJILLA CML A GRANO
       GRUESO
188 print('Histograma - INICIO')
189 hist_CG, bins_CG = np.histogram(arr_CG_states, bins = N_total_states, range
      = (0, N_total_states), density = True)
190 # Almacenamiento de histograma (probabilidades "experimentales")
191 filename_hist = './tesis_Egolf_EG_Basic_L%(length)i_G%(grainsize)i_g%(
```

```
    coupling).3f_Niter%(iterations).3e_trans%(trans).3e_Nrondas%(turns).3
    e_Nansatz%(ansatz)i_seed%(s)i_sr%(ref)i_LeastSquares10_HistDB.txt'
dict_fname = {'length': L, 'grainsize': G, 'coupling': g, 'iterations':
    N_iter, 'trans': transient, 'turns': N_turns, 'ansatz': n_tries, 's':
    seed, 'ref': sref}
np.savetxt(filename_hist % dict_fname, hist_CG)
print('Histograma - FINAL')

### DICCIONARIO CON ESTADOS S_I Y SUMAS DE PRODUCTOS ENTRE PARES DE ESPINES
    (INDEXADAS POR DISTANCIAS ENTRE SITIOS)
LCG = int(N_total_sites ** (1 / 2.0))
lattice = ini_graf(LCG)
print('Diccionario con proto-energías - INICIO')
dict_sums = {}
tenth_progress_sums = int(N_total_states / 10)
for state in range(N_total_states):
  # Marcador de progreso
  if state % tenth_progress_sums == 0:
    print('*')
  # Definición de gráfica en estado definido
  lattice = def_edo(state,lattice)
  # Cálculo de productos
  dict_prod = {}
  for site1 in range(N_total_sites - 1):
    for site2 in range(site1 + 1, N_total_sites):
      product = lattice.nodes[site1]['u'] * lattice.nodes[site2]['u']
      distance = lattice.edges[site1, site2]['path_distance']
      key = 'd' + str(float(distance))
      if key in dict_prod:
        dict_prod[key].append(product)
      else:
        dict_prod[key] = [product]
  # Cálculo de sumas de productos
  dict_sums_holder = {}
  for key in dict_prod:
    dict_sums_holder[key] = sum(dict_prod[key])
  # Definición de diccionario final
  key_state = 's' + str(float(state))
  dict_sums[key_state] = dict_sums_holder
  lattice = ini_graf(LCG)
# Almacenamiento de diccionario
filename_dictsums = './tesis_Egolf_EG_Basic_L%(length)i_G%(grainsize)i_g%(
    coupling).3f_Niter%(iterations).3e_trans%(trans).3e_Nrondas%(turns).3
    e_Nansatz%(ansatz)i_seed%(s)i_sr%(ref)i_LeastSquares10_ProtoE_dict.txt'
with open(filename_dictsums % dict_fname, "w") as write_dictsums:
  json.dump(dict_sums, write_dictsums)
print('Diccionario con proto-energías - FINAL')

### LISTA CON SUMAS DE PRODUCTOS ENTRE PARES DE ESPÍNES (PROTO-ENERGÍAS)
print('Lista con proto-energías (sin repeticiones) - INICIO')
list_sums = []
for sum_prod in dict_sums.values():
  if sum_prod not in list_sums:
    list_sums.append(sum_prod)
  else:
    continue
# Almacenamiento de lista con sumas
filename_listsums = './tesis_Egolf_EG_Basic_L%(length)i_G%(grainsize)i_g%(
```

```
    coupling).3f_Niter%(iterations).3e_trans%(trans).3e_Nrondas%(turns).3
    e_Nansatz%(ansatz)i_seed%(s)i_sr%(ref)i_LeastSquares10_ProtoE_list.txt'
with open(filename_listsums % dict_fname, "w") as write_listsums:
  json.dump(list_sums, write_listsums)
print('Lista con proto-energías (sin repeticiones) - FINAL')

### LISTA CON PROTO-ENERGÍAS, PROBABILIDADES Y NÚMERO DE ESTADOS
    CORRESPONDIENTES
print('Lista con proto-energías y probabilidades - INICIO')
protoE_desord = []
for sum_prod in list_sums:
  index_holder = []
  for key_sum in dict_sums:
    if dict_sums[key_sum] == sum_prod:
      state = int(float(key_sum[1:]))
      index_holder.append(state)
    else:
      continue
  num_states_H = len(index_holder)
  sum_probs = 0
  for index in index_holder:
    term = hist_CG[index]
    sum_probs = sum_probs + term
  protoE_desord.append([sum_prod, sum_probs, num_states_H])
# Almacenamiento de lista con probabilidades y total de estados
    correspondientes a proto-energías
filename_protoE_desord = './tesis_Egolf_EG_Basic_L%(length)i_G%(grainsize)
    i_g%(coupling).3f_Niter%(iterations).3e_trans%(trans).3e_Nrondas%(turns)
    .3e_Nansatz%(ansatz)i_seed%(s)i_sr%(ref)i_LeastSquares10_ProtoE&Prob&
    NS_desordenadas.txt'
with open(filename_protoE_desord % dict_fname, "w") as write_protoE_desord:
  json.dump(protoE_desord, write_protoE_desord)
print('Lista con probabilidades de Hamiltoniano - FINAL')

### ORDENAMIENTO DE PROTO-ENERGÍAS POR PROBABILIDAD
# Lista con probabilidades de proto-energías, ordenadas de mayor a menor
prob_ord = []
for pair in protoE_desord:
  prob_ord.append(pair[1])
prob_ord = sorted(prob_ord, reverse = True)
# Ordenamiento de lista con proto-energías y probabilidades asociadas
protoE = []
print('Ordenamiento de proto-energías por probabilidades - INICIO')
for probability in prob_ord:
  for entry in protoE_desord:
    if entry[1] == probability:
      if entry not in protoE:
        protoE.append(entry)
        break
      else:
        continue
    else:
      continue
# Almacenamiento de lista con probabilidades y total de estados
    correspondientes a proto-energías
filename_protoE = './tesis_Egolf_EG_Basic_L%(length)i_G%(grainsize)i_g%(
    coupling).3f_Niter%(iterations).3e_trans%(trans).3e_Nrondas%(turns).3
    e_Nansatz%(ansatz)i_seed%(s)i_sr%(ref)i_LeastSquares10_ProtoE&Prob&NS.txt
```

```
    '
with open(filename_protoE % dict_fname, "w") as write_protoE:
  json.dump(protoE, write_protoE)
print('Ordenamiento de proto-energías por probabilidades - FINAL')

### RECUPERACIÓN DE LISTA CON DISTANCIAS ENTRE SITIOS
# Lista con distancias
site_dists = list_dists(lattice)
print('Lista de distancias entre sitios: ')
print(site_dists)
# Lista con claves de distancias
list_key_dists_total = []
for distance in site_dists:
  list_key_dists_total.append('d' + str(distance))
print('Lista con claves de distancias entre sitios: ')
print(list_key_dists_total)



################################################################################
################################################################################
### MÍNIMOS CUADRADOS (CON ESQUEMAS LOCAL Y GLOBAL)
################################################################################
################################################################################

abs_alphas = len(list_key_dists_total)
abs_protoE = len(protoE)

### ARREGLOS - DEFAULT
# Esquema energías
final_H_def_gf = np.zeros((abs_alphas, abs_protoE))
pBoltz_def_gf = np.zeros((abs_alphas, abs_protoE))
prob_exp_def_gf = np.zeros((abs_alphas, abs_protoE))
# Esquema estado
es_def = np.zeros((abs_alphas, N_total_states))
ps_def = np.zeros((abs_alphas, N_total_states))
hCG_def = np.zeros((abs_alphas, N_total_states))
chi2s_def = np.zeros(abs_alphas)
es_def_gf = []
ps_def_gf = []
hCG_def_gf = []

### ARREGLOS - LOCAL1
# Esquema energías
final_H_l1_gf = np.zeros((abs_alphas, abs_protoE))
pBoltz_l1_gf = np.zeros((abs_alphas, abs_protoE))
prob_exp_l1_gf = np.zeros((abs_alphas, abs_protoE))
# Esquema estado
es_l1 = np.zeros((abs_alphas, N_total_states))
ps_l1 = np.zeros((abs_alphas, N_total_states))
hCG_l1 = np.zeros((abs_alphas, N_total_states))
chi2s_l1 = np.zeros(abs_alphas)
es_l1_gf = []
ps_l1_gf = []
hCG_l1_gf = []

### ARREGLOS - GRADUAL DEFAULT
# Esquema energías
```

```
final_H_grad_def_gf = np.zeros((abs_alphas, abs_protoE))
pBoltz_grad_def_gf = np.zeros((abs_alphas, abs_protoE))
prob_exp_grad_def_gf = np.zeros((abs_alphas, abs_protoE))
# Esquema estado
es_graddef = np.zeros((abs_alphas, N_total_states))
ps_graddef = np.zeros((abs_alphas, N_total_states))
hCG_graddef = np.zeros((abs_alphas, N_total_states))
chi2s_gdef = np.zeros(abs_alphas)
es_graddef_gf = []
ps_graddef_gf = []
hCG_graddef_gf = []

### ARREGLOS - GRADUAL LOCAL1
# Esquema energías
final_H_grad_l1_gf = np.zeros((abs_alphas, abs_protoE))
pBoltz_grad_l1_gf = np.zeros((abs_alphas, abs_protoE))
prob_exp_grad_l1_gf = np.zeros((abs_alphas, abs_protoE))
# Esquema estado
es_gradl1 = np.zeros((abs_alphas, N_total_states))
ps_gradl1 = np.zeros((abs_alphas, N_total_states))
hCG_gradl1 = np.zeros((abs_alphas, N_total_states))
chi2s_gl1 = np.zeros(abs_alphas)
es_gradl1_gf = []
ps_gradl1_gf = []
hCG_gradl1_gf = []


for cf in range(abs_alphas):
  # Lista reducida de distancias como claves
  list_key_dists = []
  for counter_kd, key_dist in enumerate(list_key_dists_total):
    if counter_kd > cf:
      print('Listo')
      break
    else:
      list_key_dists.append(key_dist)
  print('**********************')
  print(list_key_dists)
  print('**********************')
  # Definición de diccionario con claves para nombres de archivos
  dict_f2 = {'length': L, 'grainsize': G, 'coupling': g, 'iterations':
   N_iter, 'trans': transient, 'turns': N_turns, 'ansatz': n_tries, 's':
   seed, 'ref': sref, 'coef': cf}

  ############################ ESQUEMA LOCAL 0 (DEFAULT)
   ######################

  ### MÍNIMOS CUADRADOS - DEFAULT
  print('Mínimos cuadrados (default) - INICIO')
  ansatz_df = np.zeros((n_tries, len(list_key_dists)))
  alphas_df = np.zeros((n_tries, len(list_key_dists)))
  chi2_df = np.zeros(n_tries)
  # Minimización con múltiples ansatz
  for a_df in range(n_tries):
    # Definición de ansatz
    for value_df in range(len(list_key_dists)):
      ansatz_df[a_df][value_df] = random.uniform(-1,1)
    # Minimización
```

```
403     solution_def = minimize(min_cuadrados, ansatz_df[a_df], options={'disp':
        True})
404     alphas_df[a_df] = solution_def.x
405     chi2_df[a_df] = min_cuadrados(alphas_df[a_df])
406   # Almacenamiento
407   ansatz_df_fname = 'tesis_Egolf_EG_Chi2Std_ProbE_mAlphas_L%(length)i_G%(
        grainsize)i_g%(coupling).3f_Niter%(iterations).3e_trans%(trans).3
        e_Nrondas%(turns).3e_Nansatz%(ansatz)i_seed%(s)i_sr%(ref)i_totalcoef%(
        coef)i_LeastSquares10_Default_Ansatz.txt'
408   np.savetxt(ansatz_df_fname % dict_f2, ansatz_df)
409   alphas_df_fname = 'tesis_Egolf_EG_Chi2Std_ProbE_mAlphas_L%(length)i_G%(
        grainsize)i_g%(coupling).3f_Niter%(iterations).3e_trans%(trans).3
        e_Nrondas%(turns).3e_Nansatz%(ansatz)i_seed%(s)i_sr%(ref)i_totalcoef%(
        coef)i_LeastSquares10_Default_TotalAlphas.txt'
410   np.savetxt(alphas_df_fname % dict_f2, alphas_df)
411   chi2_df_fname = 'tesis_Egolf_EG_Chi2Std_ProbE_mAlphas_L%(length)i_G%(
        grainsize)i_g%(coupling).3f_Niter%(iterations).3e_trans%(trans).3
        e_Nrondas%(turns).3e_Nansatz%(ansatz)i_seed%(s)i_sr%(ref)i_totalcoef%(
        coef)i_LeastSquares10_Default_Chi2.txt'
412   np.savetxt(chi2_df_fname % dict_f2, chi2_df)
413   # Identificación de chi2 mínima
414   min_chi2_df = np.nanmin(chi2_df)
415   for counter_df, v_chi2_df in enumerate(chi2_df):
416     if v_chi2_df == min_chi2_df:
417       min_ansatz_df = counter_df
418   # Alphas correspondiente a mínimo
419   alphas_min_def = alphas_df[min_ansatz_df]
420   # Cálculo de probabilidades
421   pBoltz_def, Z_def = prob_boltzmann(alphas_min_def)
422   # Almacenamiento
423   fname_def = 'tesis_Egolf_EG_Chi2Std_ProbE_mAlphas_L%(length)i_G%(grainsize
        )i_g%(coupling).3f_Niter%(iterations).3e_trans%(trans).3e_Nrondas%(turns)
        .3e_Nansatz%(ansatz)i_seed%(s)i_sr%(ref)i_totalcoef%(coef)
        i_LeastSquares10_Default_ProbBoltz.txt'
424   np.savetxt(fname_def % dict_f2, pBoltz_def)
425   print('Mínimos cuadrados (local - \'trust-constr\') - FINAL')
426
427   print('Diferencia con valores finales: ' + str(min_cuadrados(
        alphas_min_def)))
428
429   ### DICCIONARIO CON COEFICIENTES ALFA FINALES - DEFAULT
430   print('Diccionario con coeficientes alfa finales (default) - INICIO')
431   dict_alphas_def = {}
432   for n_distance in range(len(list_key_dists)):
433     k_distance = list_key_dists[n_distance]
434     dict_alphas_def[k_distance] = alphas_min_def[n_distance]
435   # Almacenamiento
436   fname_alphas_def = 'tesis_Egolf_EG_Chi2Std_ProbE_mAlphas_L%(length)i_G%(
        grainsize)i_g%(coupling).3f_Niter%(iterations).3e_trans%(trans).3
        e_Nrondas%(turns).3e_Nansatz%(ansatz)i_seed%(s)i_sr%(ref)i_totalcoef%(
        coef)i_LeastSquares10_Default_Alphas.txt'
437   with open(fname_alphas_def % dict_f2, "w") as write_dict_alphas_def:
438     json.dump(dict_alphas_def, write_dict_alphas_def)
439   print('Diccionario con coeficientes alfa finales (default) - FINAL')
440
441   ### ENERGÍAS FINALES - DEFAULT
442   print('Lista con Hammiltonianos finales (default) - INICIO')
443   final_H_def = np.zeros(len(protoE))
```

```
  prob_exp = np.zeros(len(protoE))
  counter_H_def = 0
  for entry in protoE:
    H = 0
    for dist in list_key_dists:
      term = dict_alphas_def[dist] * entry[0][dist]
      H = H + term
    final_H_def[counter_H_def] = H
    prob_exp[counter_H_def] = entry[1]
    counter_H_def = counter_H_def + 1
  # Almacenamiento
  fname_H_def = 'tesis_Egolf_EG_Chi2Std_ProbE_mAlphas_L%(length)i_G%(
   grainsize)i_g%(coupling).3f_Niter%(iterations).3e_trans%(trans).3
   e_Nrondas%(turns).3e_Nansatz%(ansatz)i_seed%(s)i_sr%(ref)i_totalcoef%(
   coef)i_LeastSquares10_Default_H.txt'
  np.savetxt(fname_H_def % dict_f2, final_H_def)
  print('Lista con Hammiltonianos finales (default) - FINAL')
  # Ordenamiento
  final_H_def_gf[cf], pBoltz_def_gf[cf], prob_exp_def_gf[cf] = ordenador(
   final_H_def, pBoltz_def, prob_exp)

  ### ESQUEMA ESTADO
  es_def[cf], ps_def[cf], Zs_def = pe_estado(alphas_min_def)
  # Definición de chi2 (Estado)
  chi2s_def[cf] = np.sum((np.array(ps_def[cf]) - np.array(hist_CG)) ** 2)
  # Exclusión de probabilidades nulas
  ps_def_holder = []
  es_def_holder = []
  hCG_def_holder = []
  for counter_def, x_def in enumerate(hist_CG):
    if x_def == 0:
      continue
    else:
      ps_def_holder.append(ps_def[cf][counter_def])
      es_def_holder.append(es_def[cf][counter_def])
      hCG_def_holder.append(x_def)
  ps_def_gf.append(ps_def_holder)
  es_def_gf.append(es_def_holder)
  hCG_def_gf.append(hCG_def_holder)

  print('Z (Energía - Default): ' + str(float(Z_def)))
  print('Z (Estado - Default): ' + str(float(Zs_def)))

  ############################ ESQUEMA LOCAL 1 ('trust-constr')
   ##############

  if cf != abs_alphas + 2:
    ### MÍNIMOS CUADRADOS - LOCAL 1
    print('Mínimos cuadrados (local - \'trust-constr\') - INICIO')
    # Definición de arreglo con restricciones
    print('Definición de restricciones sobre proto-energías - INICIO')
    num_restrict = len(protoE)
    num_dists = len(list_key_dists)
    ineq_protoE = np.zeros((num_restrict, num_dists))
    for n_restriction in range(num_restrict):
      for n_distance in range(num_dists):
        key_dist = list_key_dists[n_distance]
        ineq_protoE[n_restriction][n_distance] = protoE[n_restriction][0][
```

```
    key_dist]
497     # Definición de fronteras
498     lower_limit = np.zeros(num_restrict)
499     upper_limit = np.zeros(num_restrict)
500     for indx in range(len(upper_limit)):
501       upper_limit[indx] = np.inf
502     # Definición de restricción (lineal)
503     linear_constraint = LinearConstraint(ineq_protoE, lower_limit,
    upper_limit)
504     ansatz_l1 = np.zeros((n_tries, len(list_key_dists)))
505     alphas_l1 = np.zeros((n_tries, len(list_key_dists)))
506     chi2_l1 = np.zeros(n_tries)
507     # Minimización con múltiples ansatz
508     for a_l1 in range(n_tries):
509       # Definición de ansatz
510       for value_l1 in range(len(list_key_dists)):
511         ansatz_l1[a_l1][value_l1] = random.uniform(-1,1)
512       # Minimización
513       solution_l1 = minimize(min_cuadrados, ansatz_l1[a_l1], method='trust-
    constr', constraints=linear_constraint, options={'disp': True})
514       alphas_l1[a_l1] = solution_l1.x
515       chi2_l1[a_l1] = min_cuadrados(alphas_l1[a_l1])
516     # Almacenamiento
517     ansatz_l1_fname = 'tesis_Egolf_EG_Chi2Std_ProbE_mAlphas_L%(length)i_G%(
    grainsize)i_g%(coupling).3f_Niter%(iterations).3e_trans%(trans).3
    e_Nrondas%(turns).3e_Nansatz%(ansatz)i_seed%(s)i_sr%(ref)i_totalcoef%(
    coef)i_LeastSquares10_Local1_Ansatz.txt'
518     np.savetxt(ansatz_l1_fname % dict_f2, ansatz_l1)
519     alphas_l1_fname = 'tesis_Egolf_EG_Chi2Std_ProbE_mAlphas_L%(length)i_G%(
    grainsize)i_g%(coupling).3f_Niter%(iterations).3e_trans%(trans).3
    e_Nrondas%(turns).3e_Nansatz%(ansatz)i_seed%(s)i_sr%(ref)i_totalcoef%(
    coef)i_LeastSquares10_Local1_TotalAlphas.txt'
520     np.savetxt(alphas_l1_fname % dict_f2, alphas_l1)
521     chi2_l1_fname = 'tesis_Egolf_EG_Chi2Std_ProbE_mAlphas_L%(length)i_G%(
    grainsize)i_g%(coupling).3f_Niter%(iterations).3e_trans%(trans).3
    e_Nrondas%(turns).3e_Nansatz%(ansatz)i_seed%(s)i_sr%(ref)i_totalcoef%(
    coef)i_LeastSquares10_Local1_Chi2.txt'
522     np.savetxt(chi2_l1_fname % dict_f2, chi2_l1)
523     # Identificación de chi2 mínima
524     min_chi2_l1 = np.nanmin(chi2_l1)
525     for counter_l1, v_chi2_l1 in enumerate(chi2_l1):
526       if v_chi2_l1 == min_chi2_l1:
527         min_ansatz_l1 = counter_l1
528     # Alphas correspondiente a mínimo
529     alphas_min_l1 = alphas_l1[min_ansatz_l1]
530     # Cálculo de probabilidades
531     pBoltz_l1, Z_l1 = prob_boltzmann(alphas_min_l1)
532     # Almacenamiento
533     fname_l1 = 'tesis_Egolf_EG_Chi2Std_ProbE_mAlphas_L%(length)i_G%(
    grainsize)i_g%(coupling).3f_Niter%(iterations).3e_trans%(trans).3
    e_Nrondas%(turns).3e_Nansatz%(ansatz)i_seed%(s)i_sr%(ref)i_totalcoef%(
    coef)i_LeastSquares10_Local1_ProbBoltz.txt'
534     np.savetxt(fname_l1 % dict_f2, pBoltz_l1)
535     print('Diferencia con valores finales: ' + str(min_cuadrados(
    alphas_min_l1)))
536     print('Mínimos cuadrados (local - \'trust-constr\') - FINAL')
537
538     ### DICCIONARIO CON COEFICIENTES ALFA FINALES - LOCAL 1
```

```
539     print('Diccionario con coeficientes alfa finales (local - \'trust-constr
        \') - INICIO')
540     dict_alphas_l1 = {}
541     for n_distance in range(len(list_key_dists)):
542       k_distance = list_key_dists[n_distance]
543       dict_alphas_l1[k_distance] = alphas_min_l1[n_distance]
544     # Almacenamiento
545     fname_alphas_l1 = 'tesis_Egolf_EG_Chi2Std_ProbE_mAlphas_L%(length)i_G%(
        grainsize)i_g%(coupling).3f_Niter%(iterations).3e_trans%(trans).3
        e_Nrondas%(turns).3e_Nansatz%(ansatz)i_seed%(s)i_sr%(ref)i_totalcoef%(
        coef)i_LeastSquares10_Local1_Alphas.txt'
546     with open(fname_alphas_l1 % dict_f2, "w") as write_dict_alphas_l1:
547       json.dump(dict_alphas_l1, write_dict_alphas_l1)
548     print('Diccionario con coeficientes alfa finales (local - \'trust-constr
        \') - FINAL')
549
550     ### ENERGÍAS FINALES - LOCAL 1
551     print('Lista con Hamiltonianos finales (local - \'trust-constr\') -
        INICIO')
552     final_H_l1 = np.zeros(len(protoE))
553     counter_H_l1 = 0
554     for entry in protoE:
555       H = 0
556       for dist in list_key_dists:
557         term = dict_alphas_l1[dist] * entry[0][dist]
558         H = H + term
559       final_H_l1[counter_H_l1] = H
560       counter_H_l1 = counter_H_l1 + 1
561     # Almacenamiento
562     fname_H_l1 = 'tesis_Egolf_EG_Chi2Std_ProbE_mAlphas_L%(length)i_G%(
        grainsize)i_g%(coupling).3f_Niter%(iterations).3e_trans%(trans).3
        e_Nrondas%(turns).3e_Nansatz%(ansatz)i_seed%(s)i_sr%(ref)i_totalcoef%(
        coef)i_LeastSquares10_Local1_H.txt'
563     np.savetxt(fname_H_l1 % dict_f2, final_H_l1)
564     print('Lista con Hamiltonianos finales (local - \'trust-constr\') -
        FINAL')
565     # Ordenamiento
566     final_H_l1_gf[cf], pBoltz_l1_gf[cf], prob_exp_l1_gf[cf] = ordenador(
        final_H_l1, pBoltz_l1, prob_exp)
567
568     ### ESQUEMA ESTADO
569     es_l1[cf], ps_l1[cf], Zs_l1 = pe_estado(alphas_min_l1)
570     # Definición de chi2 (Estado)
571     chi2s_l1[cf] = np.sum((np.array(ps_l1[cf]) - np.array(hist_CG)) ** 2)
572     # Exclusión de probabilidades nulas
573     ps_l1_holder = []
574     es_l1_holder = []
575     hCG_l1_holder = []
576     for counter_l1, x_l1 in enumerate(hist_CG):
577       if x_l1 == 0:
578         continue
579       else:
580         ps_l1_holder.append(ps_l1[cf][counter_l1])
581         es_l1_holder.append(es_l1[cf][counter_l1])
582         hCG_l1_holder.append(x_l1)
583     ps_l1_gf.append(ps_l1_holder)
584     es_l1_gf.append(es_l1_holder)
585     hCG_l1_gf.append(hCG_l1_holder)
```

```
586
587     print('Z (Energía - Local1): ' + str(float(Z_l1)))
588     print('Z (Estado - Local1): ' + str(float(Zs_l1)))
589
590     ########################## ESQUEMA GRADUADO 0 (DEFAULT)
      ##################
591
592   ### MÍNIMOS CUADRADOS - GRADUADO DEFAULT
593   print('Mínimos cuadrados (graduado - default) - INICIO')
594   total_alphas = len(list_key_dists)
595   ansatz_gdf = np.zeros((n_tries, len(list_key_dists)))
596   alphas_gdf = np.zeros((n_tries, len(list_key_dists)))
597   chi2_gdf = np.zeros(n_tries)
598   # Minimización con múltiples ansatz
599   for a_gdf in range(n_tries):
600     # Definición de ansatz
601     for value_gdf in range(len(list_key_dists)):
602       ansatz_gdf[a_gdf][value_gdf] = random.uniform(-1,1)
603     # Minimización gradual
604     for turn in range(total_alphas):
605       # Coeficientes fijos
606       lower_alphas = np.zeros(total_alphas)
607       upper_alphas = np.zeros(total_alphas)
608       for n_alpha in range(total_alphas):
609         if n_alpha == turn:
610           lower_alphas[n_alpha] = -np.inf
611           upper_alphas[n_alpha] = np.inf
612         else:
613           lower_alphas[n_alpha] = alphas_gdf[a_gdf][n_alpha]
614           upper_alphas[n_alpha] = alphas_gdf[a_gdf][n_alpha]
615       bounds_alphas = Bounds(lower_alphas, upper_alphas)
616       # Minimización
617       solution_grad_def = minimize(min_cuadrados, ansatz_gdf[a_gdf], options
      ={'disp': True})
618       alphas_gdf[a_gdf] = solution_grad_def.x
619       chi2_gdf[a_gdf] = min_cuadrados(alphas_gdf[a_gdf])
620   # Almacenamiento
621   ansatz_gdf_fname = 'tesis_Egolf_EG_Chi2Std_ProbE_mAlphas_L%(length)i_G%(
      grainsize)i_g%(coupling).3f_Niter%(iterations).3e_trans%(trans).3
      e_Nrondas%(turns).3e_Nansatz%(ansatz)i_seed%(s)i_sr%(ref)i_totalcoef%(
      coef)i_LeastSquares10_G_Default_Ansatz.txt'
622   np.savetxt(ansatz_gdf_fname % dict_f2, ansatz_gdf)
623   alphas_gdf_fname = 'tesis_Egolf_EG_Chi2Std_ProbE_mAlphas_L%(length)i_G%(
      grainsize)i_g%(coupling).3f_Niter%(iterations).3e_trans%(trans).3
      e_Nrondas%(turns).3e_Nansatz%(ansatz)i_seed%(s)i_sr%(ref)i_totalcoef%(
      coef)i_LeastSquares10_G_Default_TotalAlphas.txt'
624   np.savetxt(alphas_gdf_fname % dict_f2, alphas_gdf)
625   chi2_gdf_fname = 'tesis_Egolf_EG_Chi2Std_ProbE_mAlphas_L%(length)i_G%(
      grainsize)i_g%(coupling).3f_Niter%(iterations).3e_trans%(trans).3
      e_Nrondas%(turns).3e_Nansatz%(ansatz)i_seed%(s)i_sr%(ref)i_totalcoef%(
      coef)i_LeastSquares10_G_Default_Chi2.txt'
626   np.savetxt(chi2_gdf_fname % dict_f2, chi2_gdf)
627   # Identificación de chi2 mínima
628   min_chi2_gdf = np.nanmin(chi2_gdf)
629   for counter_gdf, v_chi2_gdf in enumerate(chi2_gdf):
630     if v_chi2_gdf == min_chi2_gdf:
631       min_ansatz_gdf = counter_gdf
632   # Alphas correspondiente a mínimo
```

```
633   alphas_grad_def = alphas_gdf[min_ansatz_gdf]
634   # Cálculo de probabilidades
635   pBoltz_grad_def, Z_grad_def = prob_boltzmann(alphas_grad_def)
636   # Almacenamiento
637   fname_grad_def = 'tesis_Egolf_EG_Chi2Std_ProbE_mAlphas_L%(length)i_G%(
       grainsize)i_g%(coupling).3f_Niter%(iterations).3e_trans%(trans).3
       e_Nrondas%(turns).3e_Nansatz%(ansatz)i_seed%(s)i_sr%(ref)i_totalcoef%(
       coef)i_LeastSquares10_G_Default_ProbBoltz.txt'
638   np.savetxt(fname_grad_def % dict_f2, pBoltz_grad_def)
639   print('Diferencia con valores finales: ' + str(min_cuadrados(
       alphas_grad_def)))
640   print('Mínimos cuadrados (graduado - default) - FINAL')
641
642   ### DICCIONARIO CON COEFICIENTES ALFA FINALES - GRADUADO DEFAULT
643   print('Diccionario con coeficientes alfa finales (graduado - default) -
       INICIO')
644   dict_alphas_grad_def = {}
645   for n_distance in range(total_alphas):
646     k_distance = list_key_dists[n_distance]
647     dict_alphas_grad_def[k_distance] = alphas_grad_def[n_distance]
648   # Almacenamiento
649   fname_alphas_grad_def = 'tesis_Egolf_EG_Chi2Std_ProbE_mAlphas_L%(length)
       i_G%(grainsize)i_g%(coupling).3f_Niter%(iterations).3e_trans%(trans).3
       e_Nrondas%(turns).3e_Nansatz%(ansatz)i_seed%(s)i_sr%(ref)i_totalcoef%(
       coef)i_LeastSquares10_G_Default_Alphas.txt'
650   with open(fname_alphas_grad_def % dict_f2, "w") as
       write_dict_alphas_grad_def:
651     json.dump(dict_alphas_grad_def, write_dict_alphas_grad_def)
652   print('Diccionario con coeficientes alfa finales (graduado - default) -
       FINAL')
653
654   ### ENERGÍAS FINALES - GRADUADO DEFAULT
655   print('Lista con Hammiltonianos finales (graduado - default) - INICIO')
656   final_H_grad_def = np.zeros(len(protoE))
657   counter_H_grad_def = 0
658   for entry in protoE:
659     H = 0
660     for dist in list_key_dists:
661       term = dict_alphas_grad_def[dist] * entry[0][dist]
662       H = H + term
663     final_H_grad_def[counter_H_grad_def] = H
664     counter_H_grad_def = counter_H_grad_def + 1
665   # Almacenamiento
666   fname_H_grad_def = 'tesis_Egolf_EG_Chi2Std_ProbE_mAlphas_L%(length)i_G%(
       grainsize)i_g%(coupling).3f_Niter%(iterations).3e_trans%(trans).3
       e_Nrondas%(turns).3e_Nansatz%(ansatz)i_seed%(s)i_sr%(ref)i_totalcoef%(
       coef)i_LeastSquares10_G_Default_H.txt'
667   np.savetxt(fname_H_grad_def % dict_f2, final_H_grad_def)
668   print('Lista con Hammiltonianos finales (graduado - default) - FINAL')
669   # Ordenamiento
670   final_H_grad_def_gf[cf], pBoltz_grad_def_gf[cf], prob_exp_grad_def_gf[cf]
       = ordenador(final_H_grad_def, pBoltz_grad_def, prob_exp)
671
672   ### ESQUEMA ESTADO
673   es_graddef[cf], ps_graddef[cf], Zs_grad_def = pe_estado(alphas_grad_def)
674   # Definición de chi2 (Estado)
675   chi2s_gdef[cf] = np.sum((np.array(ps_graddef[cf]) - np.array(hist_CG)) **
       2)
```

```
676   # Exclusión de probabilidades nulas
677   ps_graddef_holder = []
678   es_graddef_holder = []
679   hCG_graddef_holder = []
680   for counter_graddef, x_graddef in enumerate(hist_CG):
681     if x_graddef == 0:
682       continue
683     else:
684       ps_graddef_holder.append(ps_graddef[cf][counter_graddef])
685       es_graddef_holder.append(es_graddef[cf][counter_graddef])
686       hCG_graddef_holder.append(x_graddef)
687   ps_graddef_gf.append(ps_graddef_holder)
688   es_graddef_gf.append(es_graddef_holder)
689   hCG_graddef_gf.append(hCG_graddef_holder)
690
691   print('Z (Energía - Gradual Default): ' + str(float(Z_grad_def)))
692   print('Z (Estado - Gradual Default): ' + str(float(Zs_grad_def)))
693
694   ########################## ESQUEMA GRADUADO 1 ('trust-constr')
       ##########
695
696   if cf != abs_alphas+1:
697     ### MÍNIMOS CUADRADOS - GRADUADO L1
698     print('Mínimos cuadrados (graduado - \'trust-constr\') - INICIO')
699     # Definición de arreglo con restricciones
700     print('Definición de restricciones sobre proto-energías - INICIO')
701     num_restrict = len(protoE)
702     num_dists = len(list_key_dists)
703     ineq_protoE = np.zeros((num_restrict, num_dists))
704     for n_restriction in range(num_restrict):
705       for n_distance in range(num_dists):
706         key_dist = list_key_dists[n_distance]
707         ineq_protoE[n_restriction][n_distance] = protoE[n_restriction][0][
      key_dist]
708     # Definición de fronteras
709     lower_limit = np.zeros(num_restrict)
710     upper_limit = np.zeros(num_restrict)
711     for indx in range(len(upper_limit)):
712       upper_limit[indx] = np.inf
713     # Definición de restricción (lineal)
714     linear_constraint = LinearConstraint(ineq_protoE, lower_limit,
      upper_limit)
715     total_alphas = len(list_key_dists)
716     ansatz_gl1 = np.zeros((n_tries, len(list_key_dists)))
717     alphas_gl1 = np.zeros((n_tries, len(list_key_dists)))
718     chi2_gl1 = np.zeros(n_tries)
719     # Minimización con múltiples ansatz
720     for a_gl1 in range(n_tries):
721       # Definición de ansatz
722       for value_gl1 in range(len(list_key_dists)):
723         ansatz_gl1[a_gl1][value_gl1] = random.uniform(-1,1)
724       # Minimización gradual
725       for turn in range(total_alphas):
726         # Coeficientes fijos
727         lower_alphas = np.zeros(total_alphas)
728         upper_alphas = np.zeros(total_alphas)
729         for n_alpha in range(total_alphas):
730           if n_alpha == turn:
```

```
            lower_alphas[n_alpha] = -np.inf
            upper_alphas[n_alpha] = np.inf
          else:
            lower_alphas[n_alpha] = alphas_gl1[a_gl1][n_alpha]
            upper_alphas[n_alpha] = alphas_gl1[a_gl1][n_alpha]
        bounds_alphas = Bounds(lower_alphas, upper_alphas)
        # Minimización
        solution_grad_l1 = minimize(min_cuadrados, alphas_gl1[a_gl1], method
    ='trust-constr', constraints=linear_constraint, options={'disp': True})
        alphas_gl1[a_gl1] = solution_grad_l1.x
        chi2_gl1[a_gl1] = min_cuadrados(alphas_gl1[a_gl1])
    # Almacenamiento
    ansatz_gl1_fname = 'tesis_Egolf_EG_Chi2Std_ProbE_mAlphas_L%(length)i_G%(
    grainsize)i_g%(coupling).3f_Niter%(iterations).3e_trans%(trans).3
    e_Nrondas%(turns).3e_Nansatz%(ansatz)i_seed%(s)i_sr%(ref)i_totalcoef%(
    coef)i_LeastSquares10_G_Local1_Ansatz.txt'
    np.savetxt(ansatz_gl1_fname % dict_f2, ansatz_gl1)
    alphas_gl1_fname = 'tesis_Egolf_EG_Chi2Std_ProbE_mAlphas_L%(length)i_G%(
    grainsize)i_g%(coupling).3f_Niter%(iterations).3e_trans%(trans).3
    e_Nrondas%(turns).3e_Nansatz%(ansatz)i_seed%(s)i_sr%(ref)i_totalcoef%(
    coef)i_LeastSquares10_G_Local1_TotalAlphas.txt'
    np.savetxt(alphas_gl1_fname % dict_f2, alphas_gl1)
    chi2_gl1_fname = 'tesis_Egolf_EG_Chi2Std_ProbE_mAlphas_L%(length)i_G%(
    grainsize)i_g%(coupling).3f_Niter%(iterations).3e_trans%(trans).3
    e_Nrondas%(turns).3e_Nansatz%(ansatz)i_seed%(s)i_sr%(ref)i_totalcoef%(
    coef)i_LeastSquares10_G_Local1_Chi2.txt'
    np.savetxt(chi2_gl1_fname % dict_f2, chi2_gl1)
    # Identificación de chi2 mínima
    min_chi2_gl1 = np.nanmin(chi2_gl1)
    for counter_gl1, v_chi2_gl1 in enumerate(chi2_gl1):
      if v_chi2_gl1 == min_chi2_gl1:
        min_ansatz_gl1 = counter_gl1
    # Alphas correspondiente a mínimo
    alphas_grad_l1 = alphas_gl1[min_ansatz_gl1]
    # Cálculo de probabilidades
    pBoltz_grad_l1, Z_grad_l1 = prob_boltzmann(alphas_grad_l1)
    # Almacenamiento
    fname_grad_l1 = 'tesis_Egolf_EG_Chi2Std_ProbE_mAlphas_L%(length)i_G%(
    grainsize)i_g%(coupling).3f_Niter%(iterations).3e_trans%(trans).3
    e_Nrondas%(turns).3e_Nansatz%(ansatz)i_seed%(s)i_sr%(ref)i_totalcoef%(
    coef)i_LeastSquares10_G_Local1_ProbBoltz.txt'
    np.savetxt(fname_grad_l1 % dict_f2, pBoltz_grad_l1)
    print('Diferencia con valores finales: ' + str(min_cuadrados(
    alphas_grad_l1)))
    print('Mínimos cuadrados (graduado - default) - FINAL')

    ### DICCIONARIO CON COEFICIENTES ALFA FINALES - GRADUADO L1
    print('Diccionario con coeficientes alfa finales (graduado - \'trust-
    constr\') - INICIO')
    dict_alphas_grad_l1 = {}
    for n_distance in range(total_alphas):
      k_distance = list_key_dists[n_distance]
      dict_alphas_grad_l1[k_distance] = alphas_grad_l1[n_distance]
    # Almacenamiento
    fname_alphas_grad_l1 = 'tesis_Egolf_EG_Chi2Std_ProbE_mAlphas_L%(length)
    i_G%(grainsize)i_g%(coupling).3f_Niter%(iterations).3e_trans%(trans).3
    e_Nrondas%(turns).3e_Nansatz%(ansatz)i_seed%(s)i_sr%(ref)i_totalcoef%(
    coef)i_LeastSquares10_G_Local1_Alphas.txt'
```

```
771     with open(fname_alphas_grad_l1 % dict_f2, "w") as
    write_dict_alphas_grad_l1:
772       json.dump(dict_alphas_grad_l1, write_dict_alphas_grad_l1)
773     print('Diccionario con coeficientes alfa finales (graduado - \'trust-
    constr\') - FINAL')
774
775     ### ENERGÍAS FINALES - GRADUADO L1
776     print('Lista con Hamiltonianos finales (graduado - \'trust-constr\') -
    INICIO')
777     final_H_grad_l1 = np.zeros(len(protoE))
778     counter_H_grad_l1 = 0
779     for entry in protoE:
780       H = 0
781       for dist in list_key_dists:
782         term = dict_alphas_grad_l1[dist] * entry[0][dist]
783         H = H + term
784       final_H_grad_l1[counter_H_grad_l1] = H
785       counter_H_grad_l1 = counter_H_grad_l1 + 1
786     # Almacenamiento
787     fname_H_grad_l1 = 'tesis_Egolf_EG_Chi2Std_ProbE_mAlphas_L%(length)i_G%(
    grainsize)i_g%(coupling).3f_Niter%(iterations).3e_trans%(trans).3
    e_Nrondas%(turns).3e_Nansatz%(ansatz)i_seed%(s)i_sr%(ref)i_totalcoef%(
    coef)i_LeastSquares10_G_Local1_H.txt'
788     np.savetxt(fname_H_grad_l1 % dict_f2, final_H_grad_l1)
789     print('Lista con Hamiltonianos finales (graduado - \'trust-constr\') -
    FINAL')
790     # Ordenamiento
791     final_H_grad_l1_gf[cf], pBoltz_grad_l1_gf[cf], prob_exp_grad_l1_gf[cf] =
     ordenador(final_H_grad_l1, pBoltz_grad_l1, prob_exp)
792
793     ### ESQUEMA ESTADO
794     es_gradl1[cf], ps_gradl1[cf], Zs_grad_l1= pe_estado(alphas_grad_l1)
795     # Definición de chi2 (Estado)
796     chi2s_gl1[cf] = np.sum((np.array(ps_gradl1[cf]) - np.array(hist_CG)) **
    2)
797     # Exclusión de probabilidades nulas
798     ps_gradl1_holder = []
799     es_gradl1_holder = []
800     hCG_gradl1_holder = []
801     for counter_gradl1, x_gradl1 in enumerate(hist_CG):
802       if x_gradl1 == 0:
803         continue
804       else:
805         ps_gradl1_holder.append(ps_gradl1[cf-1][counter_gradl1])
806         es_gradl1_holder.append(es_gradl1[cf-1][counter_gradl1])
807         hCG_gradl1_holder.append(x_gradl1)
808     ps_gradl1_gf.append(ps_gradl1_holder)
809     es_gradl1_gf.append(es_gradl1_holder)
810     hCG_gradl1_gf.append(hCG_gradl1_holder)
811
812     print('Z (Energía - Gradual Local1): ' + str(float(Z_grad_l1)))
813     print('Z (Estado - Gradual Local1): ' + str(float(Zs_grad_l1)))
814
815 ##########################################################################
816
817 # Almacenamiento
818 state_p_def = 'tesis_Egolf_EG_Chi2Std_ProbE_mAlphas_L%(length)i_G%(grainsize
    )i_g%(coupling).3f_Niter%(iterations).3e_trans%(trans).3e_Nrondas%(turns)
```

```
    .3e_Nansatz%(ansatz)i_seed%(s)i_sr%(ref)
    i_LeastSquares10_Default_ProbBoltzState.txt'
819 np.savetxt(state_p_def % dict_fname, ps_def)
820 state_H_def = 'tesis_Egolf_EG_Chi2Std_ProbE_mAlphas_L%(length)i_G%(grainsize
    )i_g%(coupling).3f_Niter%(iterations).3e_trans%(trans).3e_Nrondas%(turns)
    .3e_Nansatz%(ansatz)i_seed%(s)i_sr%(ref)i_LeastSquares10_Default_HState.
    txt'
821 np.savetxt(state_H_def % dict_fname, es_def)
822
823 state_chi2_def = 'tesis_Egolf_EG_Chi2Std_ProbE_mAlphas_L%(length)i_G%(
    grainsize)i_g%(coupling).3f_Niter%(iterations).3e_trans%(trans).3
    e_Nrondas%(turns).3e_Nansatz%(ansatz)i_seed%(s)i_sr%(ref)
    i_LeastSquares10_Default_Chi2State.txt'
824 np.savetxt(state_chi2_def % dict_f2, chi2s_def)
825
826 # Almacenamiento
827 state_p_l1 = 'tesis_Egolf_EG_Chi2Std_ProbE_mAlphas_L%(length)i_G%(grainsize)
    i_g%(coupling).3f_Niter%(iterations).3e_trans%(trans).3e_Nrondas%(turns)
    .3e_Nansatz%(ansatz)i_seed%(s)i_sr%(ref)
    i_LeastSquares10_Local1_ProbBoltzState.txt'
828 np.savetxt(state_p_l1 % dict_fname, ps_l1)
829 state_H_l1 = 'tesis_Egolf_EG_Chi2Std_ProbE_mAlphas_L%(length)i_G%(grainsize)
    i_g%(coupling).3f_Niter%(iterations).3e_trans%(trans).3e_Nrondas%(turns)
    .3e_Nansatz%(ansatz)i_seed%(s)i_sr%(ref)i_LeastSquares10_Local1_HState.
    txt'
830 np.savetxt(state_H_l1 % dict_fname, es_l1)
831
832 state_chi2_l1 = 'tesis_Egolf_EG_Chi2Std_ProbE_mAlphas_L%(length)i_G%(
    grainsize)i_g%(coupling).3f_Niter%(iterations).3e_trans%(trans).3
    e_Nrondas%(turns).3e_Nansatz%(ansatz)i_seed%(s)i_sr%(ref)
    i_LeastSquares10_Local1_Chi2State.txt'
833 np.savetxt(state_chi2_l1 % dict_fname, chi2s_l1)
834
835
836
837 # Almacenamiento
838 state_p_graddef = 'tesis_Egolf_EG_Chi2Std_ProbE_mAlphas_L%(length)i_G%(
    grainsize)i_g%(coupling).3f_Niter%(iterations).3e_trans%(trans).3
    e_Nrondas%(turns).3e_Nansatz%(ansatz)i_seed%(s)i_sr%(ref)
    i_LeastSquares10_G_Default_ProbBoltzState.txt'
839 np.savetxt(state_p_graddef % dict_fname, ps_graddef)
840 state_H_graddef = 'tesis_Egolf_EG_Chi2Std_ProbE_mAlphas_L%(length)i_G%(
    grainsize)i_g%(coupling).3f_Niter%(iterations).3e_trans%(trans).3
    e_Nrondas%(turns).3e_Nansatz%(ansatz)i_seed%(s)i_sr%(ref)
    i_LeastSquares10_G_Default_HState.txt'
841 np.savetxt(state_H_graddef % dict_fname, es_graddef)
842
843 state_chi2_gdef = 'tesis_Egolf_EG_Chi2Std_ProbE_mAlphas_L%(length)i_G%(
    grainsize)i_g%(coupling).3f_Niter%(iterations).3e_trans%(trans).3
    e_Nrondas%(turns).3e_Nansatz%(ansatz)i_seed%(s)i_sr%(ref)
    i_LeastSquares10_G_Default_Chi2State.txt'
844 np.savetxt(state_chi2_gdef % dict_fname, chi2s_gdef)
845
846 # Almacenamiento
847 state_p_gradl1 = 'tesis_Egolf_EG_Chi2Std_ProbE_mAlphas_L%(length)i_G%(
    grainsize)i_g%(coupling).3f_Niter%(iterations).3e_trans%(trans).3
    e_Nrondas%(turns).3e_Nansatz%(ansatz)i_seed%(s)i_sr%(ref)
    i_LeastSquares10_G_Local1_ProbBoltzState.txt'
```

```
848 np.savetxt(state_p_gradl1 % dict_fname, ps_gradl1)
849 state_H_gradl1 = 'tesis_Egolf_EG_Chi2Std_ProbE_mAlphas_L%(length)i_G%(
    grainsize)i_g%(coupling).3f_Niter%(iterations).3e_trans%(trans).3
    e_Nrondas%(turns).3e_Nansatz%(ansatz)i_seed%(s)i_sr%(ref)
    i_LeastSquares10_G_Local1_HState.tyxt'
850 np.savetxt(state_H_gradl1 % dict_fname, es_gradl1)
851
852 state_chi2_gl1 = 'tesis_Egolf_EG_Chi2Std_ProbE_mAlphas_L%(length)i_G%(
    grainsize)i_g%(coupling).3f_Niter%(iterations).3e_trans%(trans).3
    e_Nrondas%(turns).3e_Nansatz%(ansatz)i_seed%(s)i_sr%(ref)
    i_LeastSquares10_G_Local1_Chi2State.txt'
853 np.savetxt(state_chi2_gl1 % dict_fname, chi2s_gl1)
854
855 ### GRÁFICA CON AJUSTE (Esquema Estado - 1 coeficiente)
856 plt.figure(1)
857 fig1, ax1 = plt.subplots(nrows=1, ncols=1, figsize=(10,10))
858 plt.rcParams.update({'font.size': 24})
859 plt.rcParams.update({'xtick.labelsize': 22})
860 plt.rcParams.update({'ytick.labelsize': 22})
861 plt.tight_layout(pad=4, h_pad=4, w_pad=4)
862 # Default
863 ax1.set_title(r'$N_{\alpha}=%(iter_a)i$' % {'iter_a': 1} + ', ' + r'$\chi
    ^{2}=%(chi_a)5.3e$' % {'chi_a': chi_graddef[0]})
864 ax1.set_yscale('log')
865 ax1.set_ylabel(r'$P(S_{i})$', fontsize=22)
866 ax1.set_xlabel(r'$H(S_{i})$',fontsize=22)
867 ax1.tick_params(axis='both', which='major', labelsize=20)
868 ax1.plot(es_graddef_gf[0], hCG_graddef_gf[0], 'bo', label = 'Exp',
    markeredgecolor='k', markeredgewidth=1, markersize=12)
869 ax1.plot(es_graddef_gf[0], ps_graddef_gf[0], 'r-', label = 'LS', linewidth
    =8)
870 ax1.legend(loc='lower left', fontsize=22)
871 # Almacenamiento
872 imgname1_Gis_graph = 'tesis_Egolf_EG_Chi2Std_ProbE_mAlphas_L%(length)i_G%(
    grainsize)i_g%(coupling).3f_Niter%(iterations).3e_trans%(trans).3
    e_Nrondas%(turns).3e_Nansatz%(ansatz)i_seed%(s)i_sr%(ref)
    i_LeastSquares10_GradualSelection_G_Default_GraphTotalState_Coef1.png'
873 plt.savefig(imgname1_Gis_graph % dict_fname)
874
875 ### GRÁFICA CON AJUSTE (Esquema Estado - 2 coeficientes)
876 plt.figure(2)
877 fig2, ax2 = plt.subplots(nrows=1, ncols=1, figsize=(10,10))
878 plt.rcParams.update({'font.size': 24})
879 plt.rcParams.update({'xtick.labelsize': 22})
880 plt.rcParams.update({'ytick.labelsize': 22})
881 plt.tight_layout(pad=4, h_pad=4, w_pad=4)
882 # Local 1
883 ax2.set_title(r'$N_{\alpha}=%(iter_b)i$' % {'iter_b': 1+1} + ', ' + r'$\chi
    ^{2}=%(chi_b)5.3e$' % {'chi_b': chi_graddef[1]})
884 ax2.set_yscale('log')
885 ax2.set_ylabel(r'$P(S_{i})$',fontsize=22)
886 ax2.set_xlabel(r'$H(S_{i})$',fontsize=22)
887 ax2.tick_params(axis='both', which='major', labelsize=20)
888 ax2.plot(es_graddef_gf[1], hCG_graddef_gf[1], 'bo', label = 'Exp',
    markeredgecolor='k', markeredgewidth=1, markersize=12)
889 ax2.plot(es_graddef_gf[1], ps_graddef_gf[1], 'r-', label = 'LS', linewidth
    =8)
890 ax2.legend(loc='lower left', fontsize=22)
```

```
# Almacenamiento
imgname2_Gis_graph = 'tesis_Egolf_EG_Chi2Std_ProbE_mAlphas_L%(length)i_G%(grainsize)i_g%(coupling).3f_Niter%(iterations).3e_trans%(trans).3e_Nrondas%(turns).3e_Nansatz%(ansatz)i_seed%(s)i_sr%(ref)i_LeastSquares10_GradualSelection_G_Default_GraphTotalState_Coef2.png'
plt.savefig(imgname2_Gis_graph % dict_fname)

### GRÁFICA CON AJUSTE (Esquema Estado - 3 coeficientes)
plt.figure(3)
fig3, ax3 = plt.subplots(nrows=1, ncols=1, figsize=(10,10))
plt.rcParams.update({'font.size': 24})
plt.rcParams.update({'xtick.labelsize': 22})
plt.rcParams.update({'ytick.labelsize': 22})
plt.tight_layout(pad=4, h_pad=4, w_pad=4)
# Graduado - Default
ax3.set_title(r'$N_{\alpha}=%(iter_c)i$' % {'iter_c': 2+1} + ', ' + r'$\chi^{2}=%(chi_c)5.3e$' % {'chi_c': chi_graddef[2]})
ax3.set_yscale('log')
ax3.set_ylabel(r'$P(S_{i})$', fontsize=22)
ax3.set_xlabel(r'$H(S_{i})$', fontsize=22)
ax3.tick_params(axis='both', which='major', labelsize=20)
ax3.plot(es_graddef_gf[2], hCG_graddef_gf[2], 'bo', label = 'Exp', markeredgecolor='k', markeredgewidth=1, markersize=12)
ax3.plot(es_graddef_gf[2], ps_graddef_gf[2], 'r-', label = 'LS', linewidth=8)
ax3.legend(loc='lower left', fontsize=22)
# Almacenamiento
imgname3_Gis_graph = 'tesis_Egolf_EG_Chi2Std_ProbE_mAlphas_L%(length)i_G%(grainsize)i_g%(coupling).3f_Niter%(iterations).3e_trans%(trans).3e_Nrondas%(turns).3e_Nansatz%(ansatz)i_seed%(s)i_sr%(ref)i_LeastSquares10_GradualSelection_G_Default_GraphTotalState_Coef3.png'
plt.savefig(imgname3_Gis_graph % dict_fname)

### GRÁFICA CON AJUSTE (Esquema Estado - 4 coeficientes)
plt.figure(4)
fig4, ax4 = plt.subplots(nrows=1, ncols=1, figsize=(10,10))
plt.rcParams.update({'font.size': 24})
plt.rcParams.update({'xtick.labelsize': 22})
plt.rcParams.update({'ytick.labelsize': 22})
plt.tight_layout(pad=4, h_pad=4, w_pad=4)
# Graduado - Local 1
ax4.set_title(r'$N_{\alpha}=%(iter_d)i$' % {'iter_d': 3+1} + ', ' + r'$\chi^{2}=%(chi_d)5.3e$' % {'chi_d': chi_graddef[3]})
ax4.set_yscale('log')
ax4.set_ylabel(r'$P(H_{i})$', fontsize=22)
ax4.set_xlabel(r'$H(S_{i})$', fontsize=22)
ax4.tick_params(axis='both', which='major', labelsize=20)
ax4.plot(es_graddef_gf[3], hCG_graddef_gf[3], 'bo', label = 'Exp', markeredgecolor='k', markeredgewidth=1, markersize=12)
ax4.plot(es_graddef_gf[3], ps_graddef_gf[3], 'r-', label = 'LS', linewidth=8)
ax4.legend(loc='lower left', fontsize=22)
# Almacenamiento
imgname4_Gis_graph = 'tesis_Egolf_EG_Chi2Std_ProbE_mAlphas_L%(length)i_G%(grainsize)i_g%(coupling).3f_Niter%(iterations).3e_trans%(trans).3e_Nrondas%(turns).3e_Nansatz%(ansatz)i_seed%(s)i_sr%(ref)i_LeastSquares10_GradualSelection_G_Default_GraphTotalState_Coef4.png'
plt.savefig(imgname4_Gis_graph % dict_fname)
```

```
934
935 ### GRÁFICA CON AJUSTE (Esquema Estado - 5 coeficientes)
936 plt.figure(5)
937 fig5, ax5 = plt.subplots(nrows=1, ncols=1, figsize=(10,10))
938 plt.rcParams.update({'font.size': 24})
939 plt.rcParams.update({'xtick.labelsize': 22})
940 plt.rcParams.update({'ytick.labelsize': 22})
941 plt.tight_layout(pad=4, h_pad=4, w_pad=4)
942 # Graduado - Local 1
943 ax5.set_title( r'$N_{\alpha}=%(iter_e)i$' % {'iter_e': 4+1} + ', ' + r'$\chi
    ^{2}=%(chi_e)5.3e$' % {'chi_e': chi_graddef[4]})
944 ax5.set_yscale('log')
945 ax5.set_ylabel(r'$P(H_{i})$', fontsize=22)
946 ax5.set_xlabel(r'$H(S_{i})$', fontsize=22)
947 ax5.tick_params(axis='both', which='major', labelsize=20)
948 ax5.plot(es_graddef_gf[4], ps_graddef_gf[4], 'r-', label = 'LS', linewidth
    =8)
949 ax5.plot(es_graddef_gf[4], hCG_graddef_gf[4], 'bo', label = 'Exp',
    markeredgecolor='k', markeredgewidth=1, markersize=12)
950 ax5.legend(loc='lower left', fontsize=22)
951 # Almacenamiento
952 imgname5_Gis_graph = 'tesis_Egolf_EG_Chi2Std_ProbE_mAlphas_L%(length)i_G%(
    grainsize)i_g%(coupling).3f_Niter%(iterations).3e_trans%(trans).3
    e_Nrondas%(turns).3e_Nansatz%(ansatz)i_seed%(s)i_sr%(ref)
    i_LeastSquares10_GradualSelection_G_Default_GraphTotalState_Coef5.png'
953 plt.savefig(imgname5_Gis_graph % dict_fname)
954
955 print('Programa concluido')
```

Apéndice N

Gráficas de simulaciones para prueba de distribución de Boltzmann en una rejilla de mapeo acoplado "a grano grueso" con distintos números de acoplamientos entre sitios

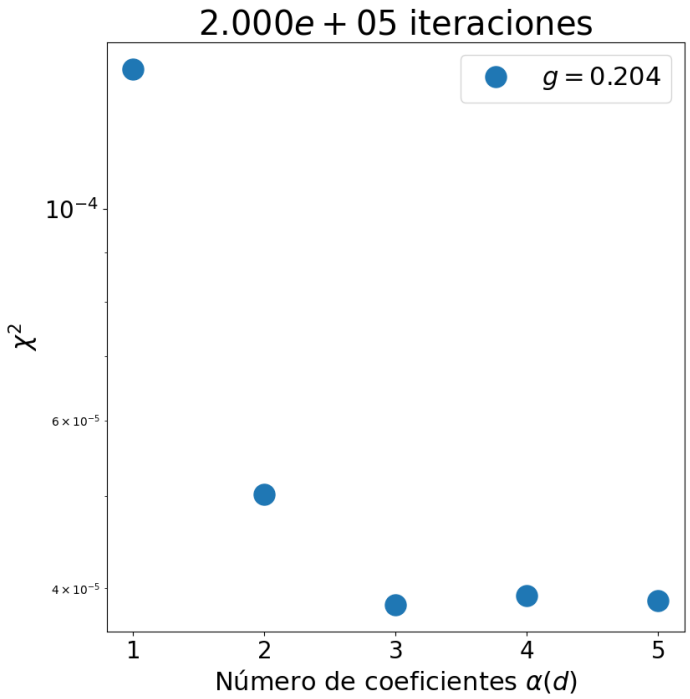
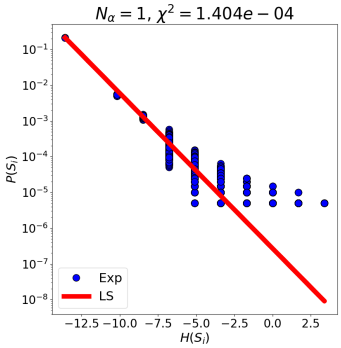
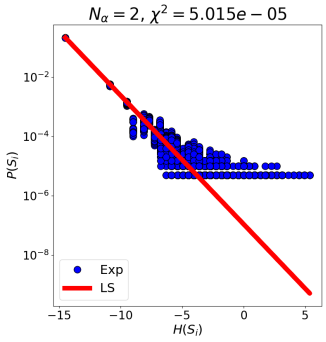


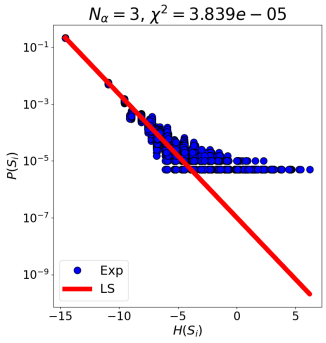
Figura N.1: Efecto de la inclusión de coeficientes $\alpha(d)$ en el cálculo de la energía $\mathcal{H}(S_i)$ sobre la dispersión χ^2 en una rejilla "a grano grueso" construida a partir de un sistema CML de longitud $L = 32$, grano $G = 8$ y acoplamiento $g0.204$. La evolución temporal del sistema CG-CML fue de 2×10^5 iteraciones.



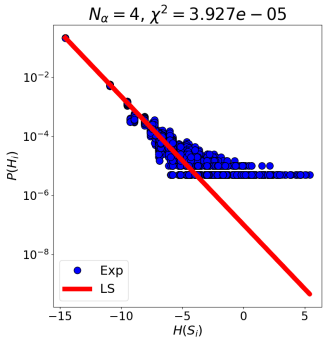
(a) Distribuciones de probabilidad de un sistema CG-CML de 4×4 sitios, considerando $N_\alpha = 1$ coeficiente para la energía $\mathcal{H}(S_i)$



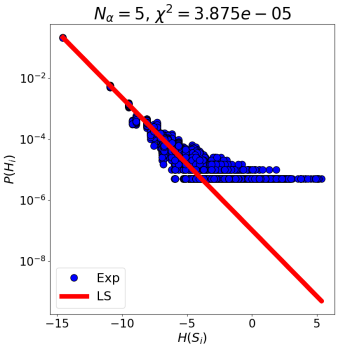
(b) Distribuciones de probabilidad de un sistema CG-CML de 4×4 sitios, considerando $N_\alpha = 2$ coeficientes para la energía $\mathcal{H}(S_i)$



(c) Distribuciones de probabilidad de un sistema CG-CML de 4×4 sitios, considerando $N_\alpha = 3$ coeficientes para la energía $\mathcal{H}(S_i)$



(d) Distribuciones de probabilidad de un sistema CG-CML de 4×4 sitios, considerando $N_\alpha = 4$ coeficientes para la energía $\mathcal{H}(S_i)$



(e) Distribuciones de probabilidad de un sistema CG-CML de 4×4 sitios, considerando $N_\alpha = 5$ coeficientes para la energía $\mathcal{H}(S_i)$

Figura N.2: Distribuciones de probabilidad $\mathcal{P}(S_i)$ obtenidas al considerar diferentes números N_α de coeficientes $\alpha(d)$ en el cálculo de la energía de un sistema CG-CML de 4×4 sitios con una evolución temporal de 2×10^5 iteraciones y semilla $s = 3316581566959966622$, derivado de una rejilla de mapeo acoplado de longitud $L = 32$, $G = 8$ y $g = 0.204$

Apéndice Ñ

Código para prueba de ergodicidad en rejilla de mapeo acoplado con acoplamientos entre múltiples vecinos

```
import sys
import random
import numpy as np
import networkx as nx
import matplotlib.pyplot as plt

##############################################################################
##############################################################################

### CÁLCULO DE DISTANCIAS ENTRE DOS ELEMENTOS DE UNA REJILLA CUADRADA
    BIDIMENSIONAL DOTADA DE CONDICIONTES DE FRONTERA PERIÓDICAS
def calc_dist(loc1, loc2, longitud):
  distx = loc1[0] - loc2[0]
  disty = loc1[1] - loc2[1]
  dist1 = (distx ** 2 + disty ** 2) ** (1 / 2)
  dist2 = ((distx - longitud) ** 2 + disty ** 2) ** (1 / 2)
  dist3 = ((distx + longitud) ** 2 + disty ** 2) ** (1 / 2)
  dist4 = (distx ** 2 + (disty - longitud) ** 2) ** (1 / 2)
  dist5 = (distx ** 2 + (disty + longitud) ** 2) ** (1 / 2)
  dist6 = ((distx + longitud) ** 2 + (disty + longitud) ** 2) ** (1 / 2)
  dist7 = ((distx - longitud) ** 2 + (disty + longitud) ** 2) ** (1 / 2)
  dist8 = ((distx - longitud) ** 2 + (disty - longitud) ** 2) ** (1 / 2)
  dist9 = ((distx + longitud) ** 2 + (disty - longitud) ** 2) ** (1 / 2)
  lista_distancias = [dist1, dist2, dist3, dist4, dist5, dist6, dist7, dist8
    , dist9]
  distancia = min(lista_distancias)
  return distancia

### INICIACIÓN DE GRÁFICA PARA SISTEMA CML
def ini_graf(longitud):
  grafica = nx.Graph()
  # Definición de nodos
  num_sitios = longitud ** 2
  for sitio in range(num_sitios):
    grafica.add_node(sitio, u = random.uniform(-1, 1))
  # Definición de bordes
  lista_edges = []
```

```
  for sitio1 in range(num_sitios):
    for sitio2 in range(num_sitios):
      if sitio2 == sitio1:
        continue
      coord1 = (sitio1 % longitud, sitio1 // longitud)
      coord2 = (sitio2 % longitud, sitio2 // longitud)
      distancia = calc_dist(coord1, coord2, longitud)
      lista_edges.append((sitio1, sitio2, {'path_distance': distancia}))
  grafica.add_edges_from(lista_edges)
  return grafica

### CÁLCULO DE NÚMERO TOTAL DE DISTANCIAS ENTRE SITIOS DE REJILLA
def total_dist(grafica):
  lista_nodos = list(grafica.nodes())
  # Lista de posibles distancias con respecto a un nodo
  lista_dists = []
  for nodo in lista_nodos:
    for vec, datos in grafica.adj[nodo].items():
      for keys, dists in datos.items():
        if dists not in lista_dists:
          lista_dists.append(dists)
  lista_dists = sorted(lista_dists)
  total_dists = len(lista_dists)
  return total_dists, lista_dists

### CÁLCULO DE N VECINOS DE UN NODO EN UNA GRÁFICA CORRESPONDIENTE A REJILLA
def n_vecinos(grafica, nodo, num_dists, lista_total_dists):
  # Lista de posibles distancias con acoplamiento no nulo
  lista_distancias = lista_total_dists[0:num_dists]
  # Listas de sitios a distancia particulares
  lista_maestra = []
  for c_dist, dist_value in enumerate(lista_distancias):
    lista_holder = []
    for vec, datos in grafica.adj[nodo].items():
      for keys, dists in datos.items():
        if dists == lista_distancias[c_dist]:
          lista_holder.append(vec)
    lista_maestra.append(lista_holder)
  return lista_maestra

### MAPEO PHI
def phi(valor_t):
  if -1 <= valor_t < -1 / 3.:
    valor_tt = (-3 * valor_t) - 2
  elif -1 / 3. <= valor_t < 1 / 3.:
    valor_tt = 3 * valor_t
  elif 1 / 3. <= valor_t <= 1:
    valor_tt = (-3 * valor_t) + 2
  return valor_tt

### EVOLUCIÓN TEMPORAL DE LA GRÁFICA
def ev_temp(num_iter, trans, x0, g_acople, grafica, lista_vecinos, guardar):
  print('INICIO DE EVOLUCION TEMPORAL')
  lista_sitios = list(grafica.nodes())
  tenth_progress = int(num_iter / 10)
  # Guardar evolución de 'u'
  if guardar == True:
    arr_promtemp = np.zeros((num_iter - trans))
```

```
94      for iteracion1 in range(num_iter):
95        if iteracion1 % tenth_progress == 0:
96          print('*')
97        # Gráfica auxiliar con valores de 'u' de siguiente iteración
98        grafica_holder1 = nx.Graph()
99        for sitio1a in lista_sitios:
100         grafica_holder1.add_node(sitio1a, u = 0)
101         # Cálculo de contribuciones asociadas a n-vecinos (por distancia)
102         sumas_dists1 = np.zeros(len(g_acople))
103         for contador_d1 in range(len(g_acople)):
104           sumvec1 = 0
105           for vecino1 in lista_vecinos[sitio1a][1][contador_d1]:
106             dif_u1 = phi(grafica.nodes[vecino1]['u']) - phi(grafica.nodes[
    sitio1a]['u'])
107           sumvec1 = sumvec1 + dif_u1
108           sumas_dists1[contador_d1] = sumvec1
109         contribuciones1 = np.dot(g_acople, sumas_dists1)
110         grafica_holder1.nodes[sitio1a]['u'] = phi(grafica.nodes[sitio1a]['u'
    ]) + contribuciones1
111       # Actualización de gráfica "original"
112       for sitio1b in lista_sitios:
113         grafica.nodes[sitio1b]['u'] = grafica_holder1.nodes[sitio1b]['u']
114       if iteracion1 <= trans - 1:
115         continue
116       arr_promtemp[iteracion1 - trans] = grafica.nodes[x0]['u']
117     print('FIN DE EVOLUCION TEMPORAL')
118     return grafica, arr_promtemp
119   # Sin guardar evolución de 'u'
120   else:
121     for iteracion2 in range(num_iter):
122       if iteracion2 % tenth_progress == 0:
123         print('*')
124       # Gráfica auxiliar
125       grafica_holder2 = nx.Graph()
126       for sitio2a in lista_sitios:
127         grafica_holder2.add_node(sitio2a, u = 0)
128         # Cálculo de contribuciones asociadas a n-vecinos (por distancia)
129         sumas_dists2 = np.zeros(len(g_acople))
130         for contador_d2 in range(len(g_acople)):
131           sumvec2 = 0
132           for vecino2 in lista_vecinos[sitio2a][1][contador_d2]:
133             dif_u2 = phi(grafica.nodes[vecino2]['u']) - phi(grafica.nodes[
    sitio2a]['u'])
134           sumvec2 = sumvec2 + dif_u2
135           sumas_dists2[contador_d2] = sumvec2
136         contribuciones2 = np.dot(g_acople, sumas_dists2)
137         grafica_holder2.nodes[sitio2a]['u'] = phi(grafica.nodes[sitio2a]['u'
    ]) + contribuciones2
138       # Actualización de gráfica
139       for sitio2b in lista_sitios:
140         grafica.nodes[sitio2b]['u'] = grafica_holder2.nodes[sitio2b]['u']
141     print('FIN DE EVOLUCION TEMPORAL')
142     return grafica
143
144 ##############################################################################
145 ##############################################################################
146
147 ### DEFINICIÓN DE PARÁMETROS DE SIMULACIÓN
```

```
148 # Selección aleatoria de semilla
149 s = random.randrange(sys.maxsize)
150 # Definición de otros parámetros
151 L = int(input('Ingresa la longitud de la rejilla CML (entero): '))
152 N_iter = int(input('Ingresa el total de iteraciones (entero): '))
153 transient = int(input('Ingresa el valor de transient (entero): '))
154 site_x0 = int(input('Ingresa un sitio de la rejilla (entero): '))
155 N_ensembles = int(input('Ingresa el total de sistemas que conforman el
        ensamble (entero): '))
156
157 print('Espera unos momentos, falta proporcionar parámetros. ')
158
159 ### GENERACIÓN DE GRÁFICA Y LISTA CON VECINOS
160 # Iniciación de generador de números aleatorios
161 random.seed(s)
162 # Definición de gráfica
163 lattice = ini_graf(L)
164 # Cálculo de distancias posibles en rejilla
165 total_dists, list_dists = total_dist(lattice)
166 print('El número total de diferentes distancias entre sitios es ' + str(
        float(total_dists)))
167 print(list_dists)
168 num_vec = int(input('Ingresa el total de acoplamientos a considerar (entero)
        : '))
169 g = np.zeros(num_vec)
170 for vec_value in range(num_vec):
171   g[vec_value] = float(input('Ingresa la constante de acoplamiento (real
        positivo) para la distancia ' + str(float(vec_value)) + ': ' ))
172 maxg = max(g)
173 ming = min(g)
174 g_fname = 'tesis_Egolf_ergodicidad_MV_L%(length)i_totalg%(coupling)i_maxg%(
        mx).3f_ming%(mn).3f_Niter%(iterations).3e_trans%(trans).3e_seed%(seed)
        i_site%(site)i_gvalues.txt'
175 g_dict = {'length': L, 'coupling': num_vec, 'mx': maxg, 'mn': ming, '
        iterations': N_iter, 'trans': transient, 'site': site_x0, 'seed': s}
176 np.savetxt(g_fname % g_dict, g)
177 # Definición de lista con n-vecinos para cada nodo
178 list_neighbors = []
179 for site in range(L**2):
180   listv = n_vecinos(lattice, site, num_vec, list_dists)
181   list_neighbors.append((site, listv))
182
183 ### DISTRIBUCIÓN DE VARIABLE 'U' EN UN SITIO PARTICULAR TRAS MÚLTIPLES
        ITERACIONES, CONSIDERANDO UN SISTEMA
184 safe_timeavg = True
185 lattice, arr_timeavg = ev_temp(N_iter, transient, site_x0, g, lattice,
        list_neighbors, safe_timeavg)
186
187 fname_timeavg = 'tesis_Egolf_ergodicidad_MV_L%(length)i_totalg%(coupling)
        i_maxg%(mx).3f_ming%(mn).3f_Niter%(iterations).3e_trans%(trans).3e_seed%(
        seed)i_site%(site)i_timeavg.txt'
188 dict_fname_timeavg = {'length': L, 'coupling': num_vec, 'mx': maxg, 'mn':
        ming, 'iterations': N_iter, 'trans': transient, 'site': site_x0, 'seed':
        s}
189 np.savetxt(fname_timeavg % dict_fname_timeavg, arr_timeavg)
190
191 print('Evolución de sistema completada')
192
```

```
193 # Distribución de variable 'u' en un sitio particular tras múltiples
      iteraciones, considerando un ensamble
194 safe_ensembleavg = True
195 arr_ensembleavg = np.zeros((N_ensembles, (N_iter - transient)))
196 for sys in range(N_ensembles):
197   lattice = ini_graf(L)
198   lattice, arr_ensemble_holder = ev_temp(N_iter, transient, site_x0, g,
      lattice, list_neighbors, safe_ensembleavg)
199   arr_ensembleavg[sys] = arr_ensemble_holder
200 arr_ensembleavg = arr_ensembleavg.flatten()
201
202 fname_ensavg = 'tesis_Egolf_ergodicidad_MV_L%(length)i_totalg%(coupling)
      i_maxg%(mx).3f_ming%(mn).3f_Niter%(iterations).3e_trans%(trans).3e_seed%(
      seed)i_site%(site)i_Nens%(ens)i_ensavg.txt'
203 dict_fname_ensavg = {'length': L, 'coupling': num_vec, 'mx': maxg, 'mn':
      ming, 'iterations': N_iter, 'trans': transient, 'site': site_x0, 'ens':
      N_ensembles, 'seed': s}
204 np.savetxt(fname_ensavg % dict_fname_ensavg, arr_ensembleavg)
205
206 print('Evolución de ensamble completada')
207
208 # Iniciación de rejilla CML para prueba de self-averaging
209 L2 = int(2*L)
210 lattice2 = ini_graf(L2)
211 # Cálculo de distancias posibles en rejilla para prueba de self-averaging
212 total_dists2, list_dists2 = total_dist(lattice2)
213 list_neighbors2 = []
214 for site2 in range(L2**2):
215   listv2 = n_vecinos(lattice2, site2, num_vec, list_dists2)
216   list_neighbors2.append((site2, listv2))
217
218 # Distribución de variable 'u' en un sistema a un tiempo fijo
219 safe_selfavg = False
220 lattice2 = ev_temp(N_iter, transient, site_x0, g, lattice2, list_neighbors2,
      safe_selfavg)
221 arr_selfavg = np.zeros(L2**2)
222 for site in range(L2**2):
223   arr_selfavg[site] = lattice2.nodes[site]['u']
224
225 fname_selfavg = 'tesis_Egolf_ergodicidad_MV_L%(length)i_totalg%(coupling)
      i_maxg%(mx).3f_ming%(mn).3f_Niter%(iterations).3e_trans%(trans).3e_seed%(
      seed)i_selfavg.txt'
226 dict_fname_selfavg = {'length': L, 'coupling': num_vec, 'mx': maxg, 'mn':
      ming, 'iterations': N_iter, 'trans': transient, 'seed': s}
227 np.savetxt(fname_selfavg % dict_fname_selfavg, arr_selfavg)
228
229 print('Prueba de self-averaging completada')
230
231 # Gráfica con resultados de ergodicidad y self-averaging
232 plt.figure(1)
233 fig1, (ax1A, ax1B, ax1C) = plt.subplots(nrows = 1, ncols = 3, figsize =
      (30,10))
234 plt.tight_layout(pad=4, h_pad=4, w_pad=6)
235
236 hist_time, bins_time = np.histogram(arr_timeavg, range = (-1, 1))
237 hist_ens, bins_ens = np.histogram(arr_ensembleavg, range = (-1, 1))
238 hist_self, bins_self = np.histogram(arr_selfavg, range = (-1,1))
239
```

```
240 ax1A.hist(bins_time[:-1], bins_time, weights = hist_time, density = True)
241 ax1A.set_title('Distribución de ' + r'$u_{\vec{x}_{0}}^{t}$' + ' con un
    sistema de %(lo)i x %(lo)i sitios ' % {'lo': L} + r'$(\vec{x}_{0} = %(
    siteref)i) $' % {'siteref': site_x0}, size=16)
242 ax1A.set_ylabel('dP(u)', size=15)
243 ax1A.set_xlabel('u', size=15)
244 for tick in ax1A.xaxis.get_major_ticks():
245   tick.label.set_fontsize(14)
246 for tick in ax1A.yaxis.get_major_ticks():
247   tick.label.set_fontsize(14)
248
249 ax1B.hist(bins_ens[:-1], bins_ens, weights = hist_ens, density = True)
250 ax1B.set_title('Distribución de ' + r'$u_{\vec{x}_{0}}^{t}$' + ' con un
    ensamble de %(Nens)i sistemas de %(lo)i x %(lo)i sitios' % {'Nens':
    N_ensembles, 'lo': L}, size=16)
251 ax1B.set_ylabel('dP(u)', size=15)
252 ax1B.set_xlabel('u', size=15)
253 for tick in ax1B.xaxis.get_major_ticks():
254   tick.label.set_fontsize(14)
255 for tick in ax1B.yaxis.get_major_ticks():
256   tick.label.set_fontsize(14)
257
258 ax1C.hist(bins_self[:-1], bins_self, weights = hist_self, density = True)
259 ax1C.set_title('Distribución de ' + r'$u_{\vec{x}}^{t_{0}}$' + ' sobre un
    sistema de %(lo)i x %(lo)i sitios ' % {'lo': L2} + r'$(t_{0} = %(tiempo)
    .2e) $' % {'tiempo': N_iter}, size=16)
260 ax1C.set_ylabel('dP(u)', size=15)
261 ax1C.set_xlabel('u', size=15)
262 for tick in ax1C.xaxis.get_major_ticks():
263   tick.label.set_fontsize(14)
264 for tick in ax1C.yaxis.get_major_ticks():
265   tick.label.set_fontsize(14)
266
267 imgname = 'tesis_Egolf_ergodicidad_MV_L%(length)i_totalg%(coupling)i_maxg%(
    mx).3f_ming%(mn).3f_Niter%(iterations).3e_trans%(trans).3e_seed%(seed)
    i_site%(site)i_Nens%(ens)i_Graph.png'
268 dict_imgname = {'length': L, 'coupling': num_vec, 'mx': maxg, 'mn': ming, '
    iterations': N_iter, 'trans': transient, 'site': site_x0, 'ens':
    N_ensembles, 'seed': s}
269 plt.savefig(imgname % dict_imgname)
270
271 print('Programa concluido')
```

Referencias bibliográficas

[1] Per Bak y Kan Chen. «Self-Organized Criticality». En: *Scientific American* 264.1 (1991), págs. 46-53.

[2] Herbert B. Callen. *Thermodynamics and an Introduction to Thermostatistics*. Singapur: John Wiley & Sons, 1985.

[3] M. C. Cross y P. C. Hohenberg. «Pattern formation outside of equilibrium». En: *Reviews of Modern Physics* 65.3 (1993), págs. 851-1123. DOI: https://doi.org/10.1103/RevModPhys.65.851.

[4] Robert L. Devaney. *A First Course in Chaotic Dynamical Systems - Theory and Experiment*. CRC Press, 2020.

[5] David A. Egolf. «Equilibrium Regained: From Nonequilibrium Chaos to Statistical Mechanics». En: *Science* 287.5450 (2000), págs. 101-104. DOI: http://dx.doi.org/10.1126/science.287.5450.101.

[6] David A. Egolf. «Far From Equilibrium». En: *Science* 296.5574 (2002), págs. 1813-1815. DOI: https://doi.org/10.1126/science.1073595.

[7] Jon Spalding Enrique Alvarez-Lacalle Blas Echebarria y Yohannes Shiferaw. «Calcium Alternans is Due to an Order-Disorder Phase Transition in Cardiac Cells». En: *Physical Review Letters* 114.108101 (2015), págs. 1-5. DOI: https://doi.org/10.1103/PhysRevLett.114.108101.

[8] J. Doyne Farmer. «Economics needs to treat the economy as a complex system». En: Paradigm Lost: Rethinking Economics and Politics (Berlín, Alemania, 12-15 de abr. de 2012).

[9] Samuel Fuenlabrada. *Probabilidad y estadística*. McGraw-Hill Interamericana, 2011.

[10] Charles Poole Herbert Goldstein y John Safko. *Classical Mechanics*. Estados Unidos de América: Addison-Wesley, 2002.

[11] Jean-Francois Rupprecht y Jacques Prost. «A fresh eye on nonequilibrium systems». En: *Science* 352.6285 (2016), págs. 514-515. DOI: https://doi.org/10.1126/science.aaf4611.

[12] Don S. Lemons. *An Introduction to Stochastic Processes in Physics*. Estados Unidos de América: The Johns Hopkins University Press, 2002.

[13] Roger Bowley y Mariana Sánchez. *Introductory Statistical Mechanics*. Estado Unidos de América: Oxford University Press, 1999.

[14] Jonathan Miller y David A. Huse. «Macroscopic equilibrium from microscopic irreversibility in a chaotic coupled-map lattice». En: *Physical Review E* 48.4 (1993).

[15] F. Reif. *Fundamentals of statistical and thermal Physics*. Tokio, Japón: McGraw-Hill, 1965.

[16] Moisés José Sametband. *Entre el orden y el caos la complejidad*. México: Fondo de Cultura Económica, 1999.

[17] Daniel V. Schroeder. *An Introduction to Thermal Physics*. Estados Unidos de América: Addison Wesley Longman, 2000.

[18] Didier Sornette. *Why Stock Markets Crash - Critical Events in Complex Financial Systems*. Estado Unidos de América: Princeton University Press, 2003.

[19] Michael Spivak. *Calculus*. España: Editorial Reverté, 2014.