Universidad de Zaragoza
Departamento de física teórica

Trabajo de fin de grado

Rejuvenecimiento y memoria en vidrios de espín desde la perspectiva de JANUS

Alejandro Clavero Álvarez

Director: Sergio Perez Gaviro
Ponente: Alfonso Tarancón Lafita

A Sara

Índice general

Capítulo 1

Introducción

Los vidrios de espín (*spin glasses*) son materiales caracterizados por el inusual comportamiento de sus momentos magnéticos. En lugar de mostrar un alineamiento regular de éstos, como los materiales ferromagnéticos o los antiferromagnéticos, los momentos están distribuidos de forma aleatoria, de forma análoga a como la red atómica de un vidrio está dispuesta de forma irregular frente a la estructura regular de un cristal.

Por debajo de una temperatura crítica T_C estos materiales abandonan la fase paramagnética para pasar a una fase conocida como *fase vidrio de espín*, en la cual se dan los efectos que se estudian en esta memoria: la *memoria* y el *rejuvenecimiento*. Para entender dichos efectos, es especialmente ilustrativo considerar dos tipos de experimento (figura 1.1).

En el *dip protocol* [1] medimos la curva de la parte imaginaria de la susceptibilidad χ'' en función de la temperatura de tres formas distintas.

En primer lugar, recorremos el rango de temperaturas enfriando o calentando el sistema a un ritmo constante, obteniendo la curva de referencia. En segundo lugar, medimos enfriando el sistema a un ratio determinado, pero cesando el enfriamiento por un momento a una temperatura $T = 12\,\mathrm{K}$ y retomándolo después; durante el tiempo que se deja reposar al sistema a temperatura constante χ'' decae, efecto conocido como envejecimiento. Por último, volvemos a medir $\chi''(T)$ esta vez calentando el sistema a un ratio determinado y sin detenernos en $T = 12\,\mathrm{K}$. Sorprendentemente, el sistema "recuerda" (efecto de memoria) que su enfriamiento se cesó a $T = 12\,\mathrm{K}$ y en lugar de seguir la curva de referencia medida, sigue la de enfriamiento, volviendo a decrecer su susceptibilidad en $T = 12\,\mathrm{K}$ la misma cantidad que decreció en el enfriamiento.

En el protocolo de dos temperaturas [2] comenzamos con un vidrio de espín a temperatura $T_1 = 12\,\mathrm{K}$. Dejamos evolucionar al sistema a temperatura constante, y vemos que la parte imaginaria de la susceptibilidad χ'' decae de forma continua, efecto al que llamamos envejecimiento. A continuación, cambiamos a una temperatura más baja $T_2 = 9\,\mathrm{K}$. χ'' experimenta un crecimiento súbito, efecto al que en contraposición con el envejecimiento se le llama rejuvenecimiento. Por último, tras dejar decrecer a χ'' por un tiempo a T_2, volvemos a calentar al sistema a T_1. El sistema recupera de forma rápida la χ'' que tenía cuando se pasó de T_1 a T_2, y continua la evolución como si nunca hubiera existido la fase a T_2, con la misma pendiente.

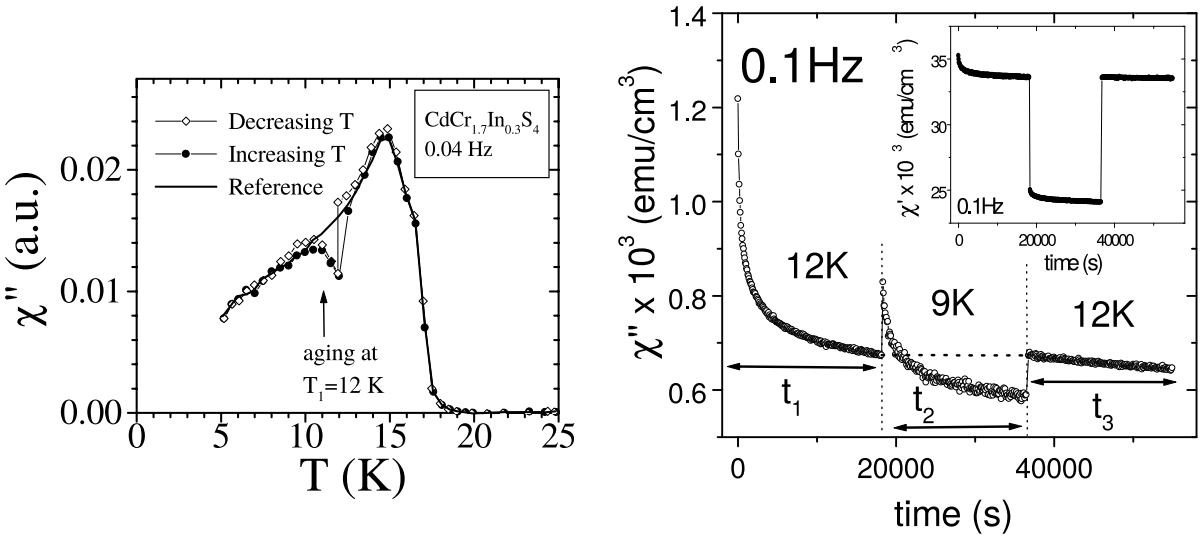


Figura 1.1: Experimentos que muestran los efectos de memoria y rejuvenecimiento en un vidrio de espín de $CdCr_{1.7}In_{0.3}S_4$ para la parte imaginaria χ'' y real χ' de la susceptibilidad. En la figura izquierda (tomada de la figura 1 de [1]) vemos el dip protocol y en la derecha (tomada de la figura 1 de [2]) el protocolo de dos temperaturas.

Modelo

Los primeros vidrios de espín [3] que se estudiaron en los años 70 consistían en metales nobles impurificados con metales de transición. Las impurezas crean perturbaciones aleatorias en la red cristalina modelables mediante el *modelo RKKY*, que propone una interacción de canje entre dos momentos magnéticos a distancia r de la forma

$$J(r) \propto \frac{\cos(2K_F r)}{r^3} \tag{1.1}$$

donde $K_F \sim 1 \times 10^{10}\,\mathrm{m}^{-1}$ es el vector de ondas del metal impurificado. La función $J(r)$ tiene una oscilación debida al término coseno de longitud de onda muy corta, cambiando de signo a escala atómica.

Los momentos magnéticos de los vidrios de espín se modelarán mediante espines discretos situados en una malla tridimensional de L espines de lado con espaciado regular, como propusieron Edwards y Anderson [4]. Supondremos únicamente interacciones a primeros vecinos (figura 1.2) y condiciones de contorno periódicas, de forma que cada espín s_i interactúe con sus seis vecinos s_j mediante un hamiltoniano

$$\mathscr{H} = -\sum_{\langle i,j \rangle} J_{ij} s_i s_j \tag{1.2}$$

donde los acoplos $J_{ij} \in \pm 1$ de los espines $s_i \in \pm 1$ son números aleatorios de distribución plana, modelizando la rápida variación de J debida a la interacción RKKY. La función de partición del sistema queda como

$$\mathscr{Z} = [Z_J] \tag{1.3}$$

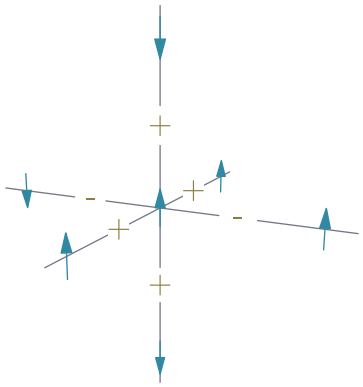


Figura 1.2: Modelo de Edwards-Anderson en 3D, con interacciones a primeros vecinos. Cada espín interacciona únicamente con los 6 espines más cercanos en la red, siendo el signo de la interacción aleatorio.

donde "[ ]" indica un promedio sobre sistemas con diferentes acoplos J_{ij}, a los que llamaremos *muestras*. Este promediado esta justificado por las escalas de tiempos existentes en el problema: los espines s_i tienen una evolución mucho más rápida que los acoplos, de forma los acoplos se pueden suponer fijos durante las iteraciones que se pueden realizar en la simulación de los sistemas. Los promedios usuales sobre el espacio de configuraciones de una sola muestra se denotan como "$\langle\ \rangle$".

Magnitudes

Para estudiar la memoria y el rejuvenecimiento, se definieron y midieron diversos observables del sistema. Es especialmente interesante el comportamiento de estas magnitudes en el protocolo de dos de temperaturas al que se someterá a los vidrios de espín,

$$T_1 \longrightarrow T_2 \longrightarrow T_1$$

donde $T_2 < T_1 < T_C$ y los cambios de T_1 a T_2 y de T_2 a T_1 se producen en t_w^1 y t_w^2 respectivamente.

La energía, de forma similar al hamiltoniano del sistema, se define como

$$E = -\frac{1}{V}\sum_{\langle i,j\rangle} J_{ij}s_i s_j \tag{1.4}$$

con $V = L^{-3}$ y tomando valores entre -1 y 1, es indispensable para actualizar los espines en cada iteración del algoritmo de Metropolis-Hasting. Tiene un comportamiento suave, lo que la hace útil para comprobar si la simulación ha transcurrido de forma normal, los datos no se han corrompido al guardarlos, etc.

En los sistemas ferromagnéticos es de especial interés la magnetización del sistema, aquí definida como $\frac{1}{V}\sum_i \sigma_i$, por ser el parámetro de orden. No obstante, los vidrios de espín tienen una magnetización global nula, de forma que es necesario definir como parámetro de orden el *overlap* entre los espines s_i^a, s_i^b de dos réplicas a y b como

$$q = \frac{1}{V}\sum_i s_i^a, s_i^b \tag{1.5}$$

donde se define como pareja de réplicas a dos sistemas con los mismos acoplos J_{ij} pero que han evolucionado de forma independiente, con distintos números aleatorios en el proceso. Es una magnitud comprendida entre −1 y 1, que para tiempos mayores a las 10^8 iteraciones de Montecarlo fluctúa alrededor de $q = 0$ con media nula, como puede verse más adelante en la figura 3.2. La distribución del *overlap* es objeto de debate, habiéndose propuesto una forma de doble pico con o sin un continuo de estados en medio.

La correlación a dos tiempos t_w, $t_w + t_0$ se define como

$$C(t_w, t_w + t_0) = \frac{1}{V} \sum_i \langle s_i(t_w)\ s_i(t_w + t_0) \rangle \tag{1.6}$$

donde $s_i \in$ son los V espines del sistema. El parámetro t_0 está relacionado con la frecuencia experimental de medición de la susceptibilidad ω como $t_0 \simeq 2\pi/\omega$.

La magnitud en la que se ven con mayor claridad los efectos de memoria y rejuvenecimiento es la susceptibilidad χ, que se puede calcular gracias al teorema de fluctuación-disipación a través de la correlación como

$$\chi(t_w, t_w + t_0) = \beta(1 - C) \tag{1.7}$$

donde $\beta = 1/k_B T$. Si bien este resultado sólo es válido en equilibrio térmico, condición extremadamente difícil de alcanzar en sistemas con alta frustración como los vidrios de espín, se puede extender fácilmente fuera de él multiplicando por una función X que sólo depende de los tiempos t_1, t_2 empleados en la correlación. En este trabajo, no se ha realizado dicha corrección por estar fuera de los objetivos fijados y requerir un análisis más profundo de estos sistemas.

Por último, mediante el *overlap* se puede definir la correlación espacial C_4 como

$$C_4(r, t_w) = \sum_i \langle q_i(t_w)\ q_{i+r}(t_w) \rangle \tag{1.8}$$

donde q_{i+r} es el *overlap* del espín que está a distancia r del i. La magnitud, con valores entre −1 y 1, aumenta conforme el sistema se acerca al equilibrio ($t_w \longrightarrow \infty$) y disminuye con la distancia entre espines r.

Capítulo 2

Objetivos

El principal objetivo del trabajo presentado es caracterizar los efectos de memoria y rejuvenecimiento en vidrios de espín. Para ello ha sido necesario familiarizarse con los vidrios de espín, paradigma de los sistemas complejos, y paralelamente realizar una inmersión a fondo en la supercomputación, utilizando diferentes arquitecturas (PC, *clusters*, y ordenadores de propósito específico) y lenguajes de programación (Python, C99, Julia [5] y Unix Shell) con los que crear código que realizara y analizara las simulaciones.

Definiendo la energía de los estados como el valor del hamiltoniano se puede explorar el espacio de configuraciones[1] y asignándoles probabilidades $e^{-\beta\mathscr{H}}$ con $\beta = 1/k_B T$ las magnitudes del sistema introducidas pueden aproximarse mediante un algoritmo como el de Metrópolis-Hasting [6]: es simple de implementar, acerca al sistema a la termalización antes que otros algoritmos como el *Heat bath* y es muy eficiente computacionalmente. El algoritmo requiere una gran cantidad de numeros aleatorios, cuya calidad se comprobó exhaustivamente (Apéndice B). Cada iteración del algoritmo equivale a aproximadamente un picosegundo [7], lo que limita los intervalos temporales en el estudio de estos sistemas a las décimas de segundo.

Los cálculos se realizaron en el *cluster* Memento y en el ordenador de propósito específico Janus [8], basado en FPGAs[2]. Para calcular los observables es necesario un gran número de muestras, lo que se consiguió de forma diferente en cada arquitectura:

- En el *cluster* el gran número de nodos de computación, los múltiples hilos de ejecución por cada procesador, y el *multispin coding* [9] (Apéndice C) permiten simular miles de muestras simultáneamente.

- En Janus, sólo es posible simular del orden de una centena de vidrios de espín simultáneamente. No obstante, se utilizó para retículos grandes ($L > 40$), en los que los efectos de tamaño finito son mucho menos notables y las fluctuaciones de los observables son

[1] Temperaturas demasiado bajas no son explorables con este método. Valores muy altos para $\beta = 1/k_B T$ crean un *underflow* en el cálculo de $\exp(-\beta E)$, es decir, el procesador no puede distinguir $\exp(-\beta E)$ de cero. No obstante, se ha comprobado que el límite está en $T = 1/60$, muy inferior a las temperaturas usadas.

[2] Un FPGA (del inglés *Field Programable Gate Array*) es un dispositivo basado en puertas lógicas que permite programar las conexiones entre sus puertas internas mediante un ordenador. Esto permite crear circuitos electrónicos adaptados a la aplicación, consiguiendo altas velocidades de cálculo.

menores, requiriendo promedios entre muchas menos muestras (decenas) para conseguir la misma incertidumbre en las medidas.

Tanto Memento como Janus resultan ineficaces para realizar simulaciones pequeñas, por lo que éstas se realizaron en un PC doméstico.

Se decidió realizar todas las simulaciones bajo un protocolo de dos temperaturas como el de la figura 1.1:

1. El sistema, de un L determinado, se deja evolucionar t_w^1 pasos de Montecarlo a una temperatura T_1.

2. Una vez realizados los t_w^1 pasos de Montecarlo, se enfría de forma instantánea a $T_2 < T_1$. En este cambio se medirá el efecto del rejuvenecimiento. El sistema permanece a T_2 hasta que se alcance el paso de Montecarlo t_w^2.

3. Alcanzado dicho paso, se vuelve a calentar instantáneamente el sistema a T_1. Se le deja permanecer a esta temperatura 10^8 iteraciones para poder observar la memoria del sistema.

Además, hay realizar una simulación de referencia a $T = T_1 = \text{cte.}$ con la que comparar la susceptibilidad para medir rejuvenecimiento y memoria.

En dicho protocolo, se simuló $t_w^1 \in \{10^4, 10^5, 10^6, 10^7, 10^8\}$, $T_1 \in \{0.9, 0.8, 0.7, 0.6\}$, $T_2 \in \{0.8, 0.7.0.6\}$ y se empleó $t_0 \in \{10^4, 10^5, 10^6, 10^7, 10^8\}$. Se emplearon varios tamaños L de retículo para la simulación de referencia, $L \in \{4, 6, 8, 12\}$ en Memento y $L \in \{8, 48, 80\}$ en Janus. El protocolo de dos temperaturas pudo completarse hasta t_w^2 para $L \in \{8, 12, 48, 80\}$ y más allá para $L \in \{8, 48, 80\}$.

Como valor para t_w^2, se escogió $t_w^1 + 10^8$ para las redes de Memento y 2×10^8 para las redes de Janus.

Capítulo 3

Desarrollo

Los tiempos de simulación y medición de magnitudes llegan a la semana para algunos tamaños de sistema, por lo que es imprescindible una buena organización a la hora de programar las simulaciones y las medidas. Además, es importante comprobar que los sistemas están evolucionando como se espera, sin fallos de *software* o *hardware* que invaliden los resultados. Por ello, mientras se obtenían las configuraciones de espines, se comprobó que todos los observables del sistema tenían el comportamiento esperado.

Comportamiento de los observables

Un observable especialmente útil para comprobar que no ha habido fallos graves en la simulación es la energía, que ha de tener un comportamiento suave y continuo como el que muestra la figura 3.1. El *overlap*, como muestra la figura 3.2, debe tener un histograma centrado en cero y una distribución de doble pico simétrica.

Como puede verse en la figura 3.3, los sistemas están más descorrelacionados cuanto más aumentamos el intervalo temporal t_0 debido a que se comparan configuraciones de espines separadas por un mayor tiempo, y van alcanzando un estado global de mayor correlación conforme lo dejamos evolucionar ($t_w \longrightarrow \infty$) y el sistema se acerca lentamente al equilibrio térmico.

La dependencia de la correlación con L es mucho más marcada para t_0 elevado, como puede

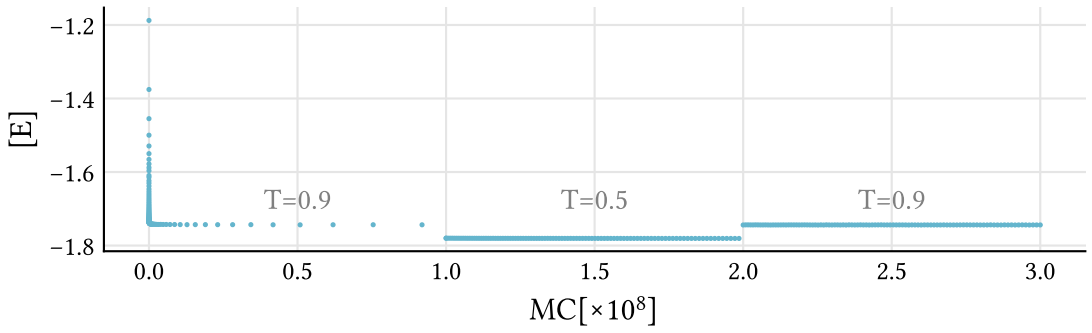


Figura 3.1: Energía para un sistema $L = 80$ promediada sobre 8 muestras.

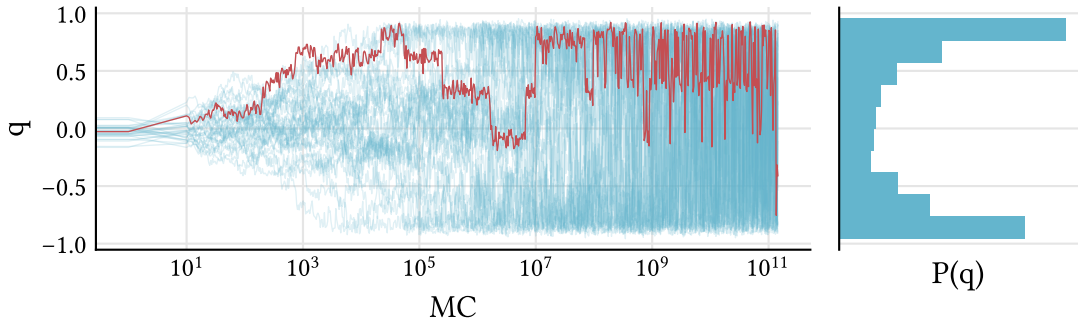


Figura 3.2: Fluctuaciones del overlap q para L = 8, uno de los pocos tamaños que puede llevarse hasta 10^{11} iteraciones de Montecarlo en un tiempo de días. Se muestran en el fondo (azul) el overlap de 32 muestras y en rojo el de una de ellas. A la derecha se muestra un histograma de la distribución del overlap de las muestras por encima de 10^8 pasos de Montecarlo.

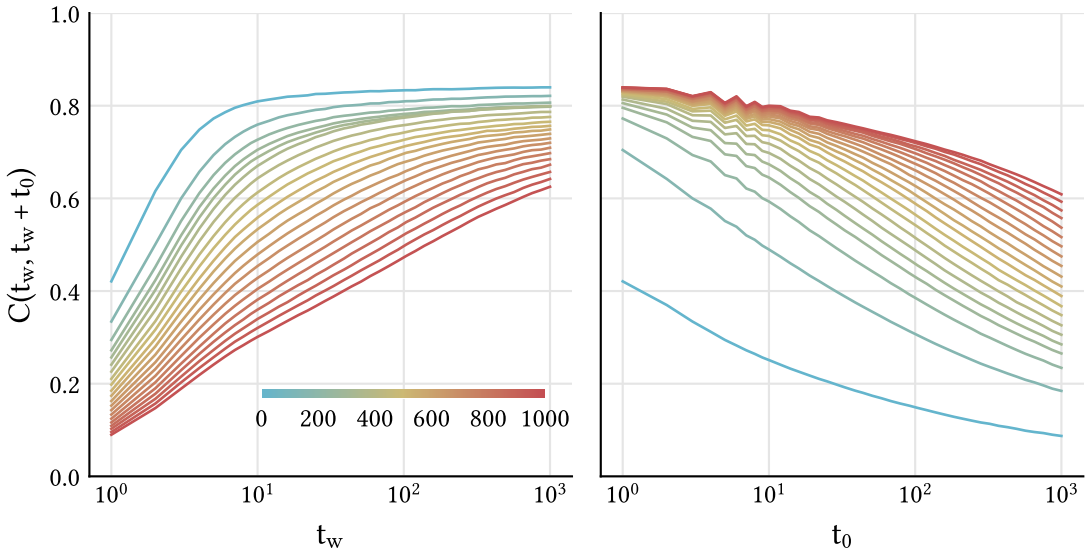


Figura 3.3: Correlación para un sistema L = 32 promediada sobre 210 muestras. Se muestra en la escala de color el parámetro (t_w o t_0) no empleado en el eje horizontal. Ambos parámetros toman valores $t_0, t_w \in [10^0, 10^{0.1}, 10^{0.2}, \ldots, 10^{2.9}, 10^3]$.

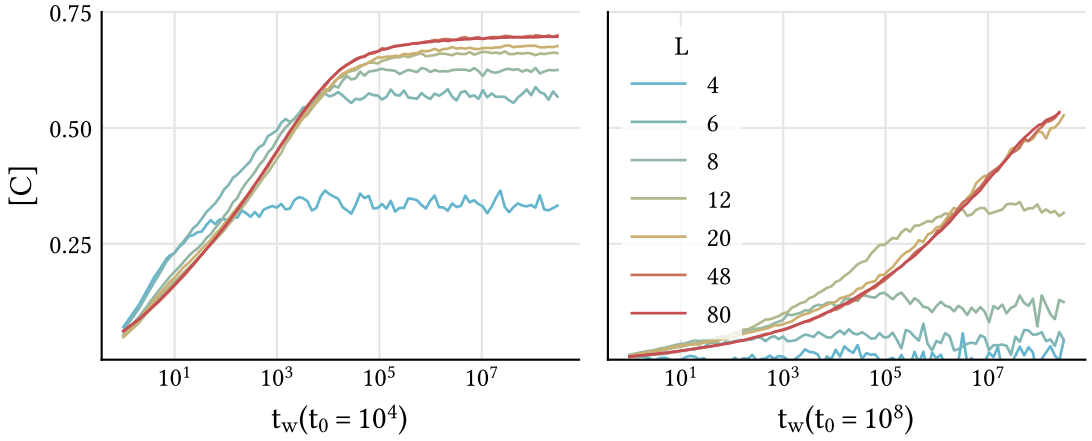


Figura 3.4: Correlación a $T = 0.9$ promediada sobre más de 1000 muestras para $L < 40$ y sobre más de 30 para $L > 20$ (color en el original). Para t_0 y t_w bajos las medidas de la correlación están menos afectadas por la L del sistema.

verse en la figura 3.4. También vemos que de forma cualitativa la correlación crece monótonamente con L para $t_w \longrightarrow \infty$.

La correlación espacial es especialmente interesante para saber si la muestra va a mostrar unos efectos de tamaño finito fuertes, ya que con ella se puede hallar una longitud de correlación ξ mediante el ajuste asintótico [10]

$$C_4(r, t_w) \overset{L\to\infty}{=} \frac{A}{r^\alpha} \exp\left([-r/\xi(t_w)]^\beta\right) \tag{3.1}$$

o mediante integración [11] de C_4 con

$$\xi(t_w) \simeq \frac{\int_0^{L/2} r^2 C_4(r, t_w)\,\mathrm{d}r}{\int_0^{L/2} r C_4(r, t_w)\,\mathrm{d}r} \tag{3.2}$$

Nótese que no tiene sentido ver el valor de C_4 para $r > L/2$, ya que las condiciones de contorno periódicas implican que la distancia entre dos espines es siempre igual o menor que ese valor. Una longitud de correlación cercana a este valor es señal de que el retículo es demasiado pequeño como para considerarlo.

Se consideró cuál de los métodos para hallar ξ emplear; en la figura 3.5 se puede ver un ejemplo de los resultados de ambos para un retículo con $L = 12$. Se escogió el método mediante integrales de la ecuación (3.2) por ser mucho más preciso para $t_w \gg 10^7$ que el ajuste a la ecuación 3.1.

En la figura 3.6 se muestran los resultados obtenidos para la longitud de correlación. Como se puede ver, siempre se tiene $\xi \ll L/2$, tal y como se esperaba.

Rejuvenecimiento

Antes de estudiar el rejuvenecimiento es importante encontrar una definición cuantitativa; en la figura 3.7 se exponen las posibles definiciones creadas para el protocolo de dos temperaturas.

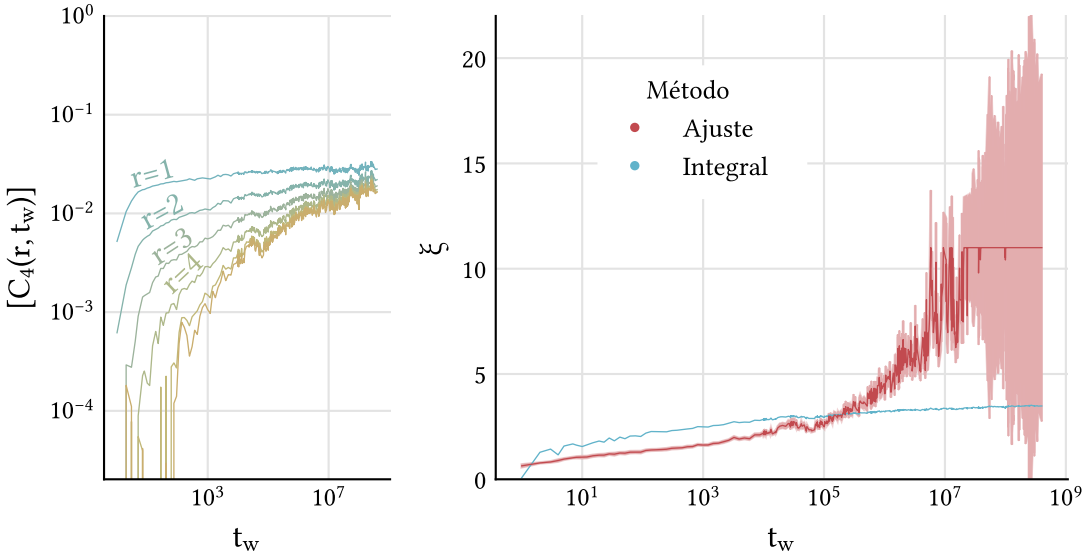


Figura 3.5: En la figura izquierda se muestran los valores de C_4 medidos para un retículo $L = 12$ a $T = 0.9$, correspondiendo cada línea a un $r \in \{1, 2, \dots, 6\}$ junto a los ξ obtenidos mediante el ajuste y mediante integración en la figura derecha (color en el original). Para t_w alta el ajuste de C_4 es malo y la incertidumbre (zona sombreada) crece sin límite.

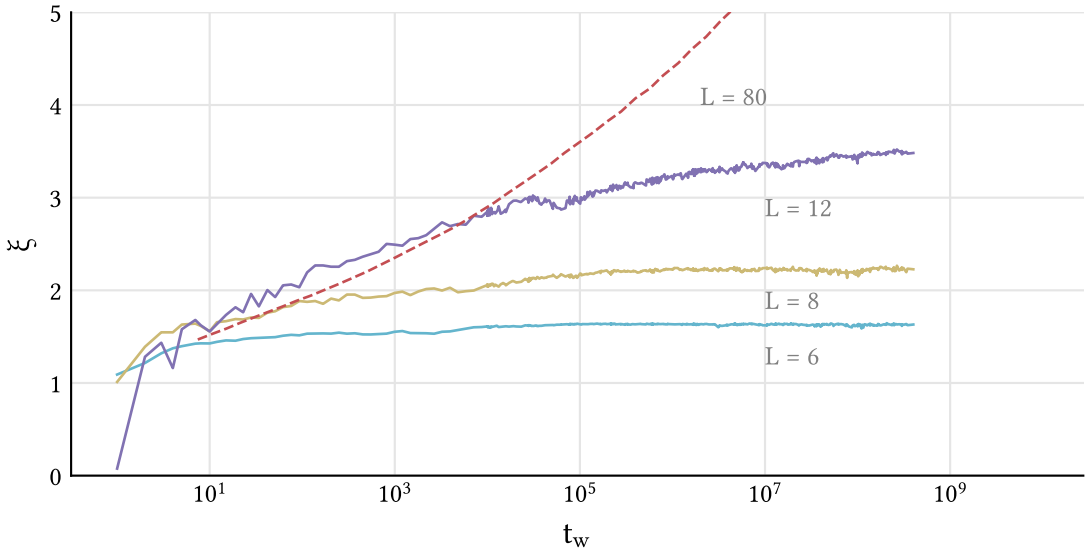


Figura 3.6: La longitud de correlación ξ a $T = 0.7$ es menor que $L/2$ en todo el rango de t_w empleado; la imagen no cambia sustancialmente para otras temperaturas $T < T_C$. Los datos para $L = 80$ se han extraído de [11].

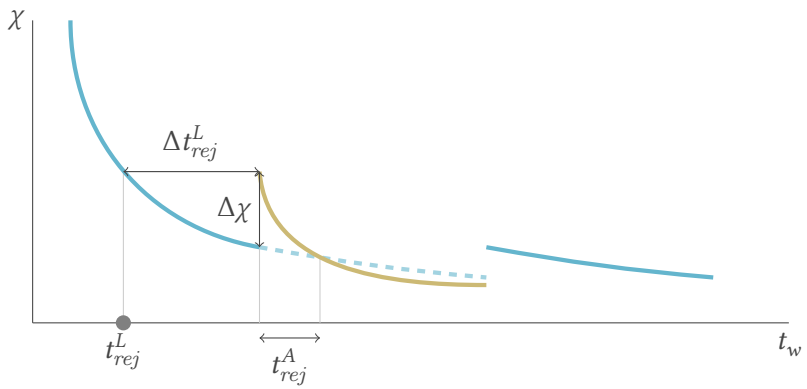


Figura 3.7: Definiciones cuantitativas del rejuvenecimiento en un protocolo de dos temperaturas. Se muestra la susceptibilidad $\chi(t_w)$.

Hasta ahora sólo se han visto ejemplos de rejuvenecimiento *fuerte*, en el que la susceptibilidad sube por encima del valor que tenía en T_1 al cambiar a T_2. No obstante, en algunas de las mediciones se detectó rejuvenecimiento *débil*, en el que la curva de χ no sube por encima de T_1 al efectuar el cambio a T_2. $\Delta\chi$ es la única magnitud que no diverge en esos casos, por lo que se usará en el análisis. Para el caso del rejuvenecimiento débil tendrá valores negativos.

Se realizó un *bootstrap* [12] (Apéndice A) de $\chi(t_w)$ de las muestras de forma que se tuviera una incertidumbre para los $\Delta\chi$ hallados, que se calcularán restando la χ de referencia a T_1 y la χ de la fase a T_2 en $t_w = t^1_w$. El número de muestras tomadas varía según los parámetros t, t^1_w y $\frac{T_1-T_2}{T_1}$, pero se encuentra siempre entre 15 y 30 para los retículos $L > 40$ y es superior a mil para los $L < 40$. Además, por cada muestra se promediaron dos réplicas.

Memoria

Antes de analizar los datos tomados, se realizó una pequeña simulación con t_w bajo y gran cantidad de estadística para ver cómo se manifestaba la memoria en un caso ideal (figura 3.8) en la que se aprecia como existe un pequeño tiempo en el que el sistema todavía no ha recuperado la curva de referencia. Podemos caracterizar ese tiempo Δt viendo cuantas iteraciones pasan hasta que ambas curvas se cruzan, obteniéndose una magnitud escalar para la memoria.

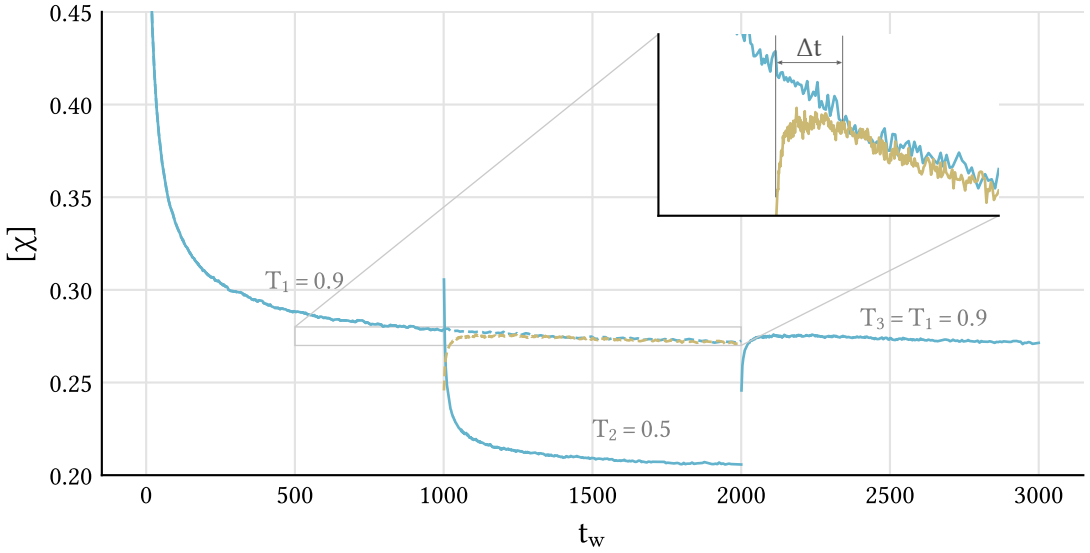


Figura 3.8: Tras volver a T_1 desde T_2 el sistema (promedio sobre 10500 muestras, $L = 12$) recupera la curva $\chi(T_1)$ desde donde se produjo el cambio a T_2. Se muestran en línea discontinua azul una simulación a T_1 sin cambio a T_2 como referencia, y en línea discontinua amarilla la curva para T_3 desplazada a la izquierda para facilitar la comparación.

Capítulo 4

Resultados

Una vez hallados los $\Delta\chi$ en t_w^1 podemos analizar el efecto del rejuvenecimiento, y con los Δt hasta que el sistema recupera la memoria podemos cuantificar ésta en función de los parámetros.

Rejuvenecimiento

Se comprobó si el parámetro t_0 de la correlación jugaba un papel importante en la aparición de rejuvenecimiento débil. Como puede verse en la figura 4.1, en la que se realizó una simulación para baja t_w, aumentar t_0 hace disminuir $\Delta\chi$ de forma consistente hasta que desaparece el rejuvenecimiento fuerte para $t_0 \sim \frac{1}{2} t_w^1$. No obstante, hay que notar que experimentalmente $\omega t_w > 1$, por lo que $t_0 < 2\pi t_w$, lo que restringe el rango de t_0.

En la figura 4.2, puede verse que los parámetros de los que depende el rejuvenecimiento (t_w^1, t_0, $\frac{T_1 - T_2}{T_1}$) tienen una influencia distinta sobre $\Delta\chi$.

Notamos que el valor de $\Delta\chi$ parece tener un mínimo en t_0, visible únicamente para $t_w^1 < 10^5$. Además, hay un cambio en el comportamiento de $\Delta\chi$ conforme aumentamos t_0, que corresponde experimentalmente a disminuir la frecuencia de medición de la susceptibilidad: para t_0 bajo, una mayor t_w^1 implica un mayor rejuvenecimiento $\Delta\chi$, mientras que para t_0 alto (rango inaccesible experimentalmente) ocurre al revés, y menores t_w^1 exhiben mayor $\Delta\chi$. Como predecía el pequeño experimento de la figura 4.1, las redes que muestran rejuvenecimiento fuerte pasan a mostrar rejuvenecimiento débil al aumentar t_0.

La influencia de $T_1 - T_2$ no es muy clara para $L > 8$, pero para $L = 8$ parece que aumentarla hace disminuir $\Delta\chi$. En todo caso, es el parámetro que menos influye en el rejuvenecimiento, probablemente por ser el rango de valores tomados en el resto de parámetros mucho mayor.

Por último, notamos que sólo se ha obtenido rejuvenecimiento fuerte para $L = 48$ y $L = 80$, lo que indica que el tamaño de la red es relevante en la aparición de este efecto.

Memoria

Para explorar la forma funcional de Δt, se optó por representarlo en función de sus parámetros t_w^1, t_0 y $\frac{T_2 - T_1}{T_1}$ promediando sobre el resto, de forma que pudieran verse tendencias generales

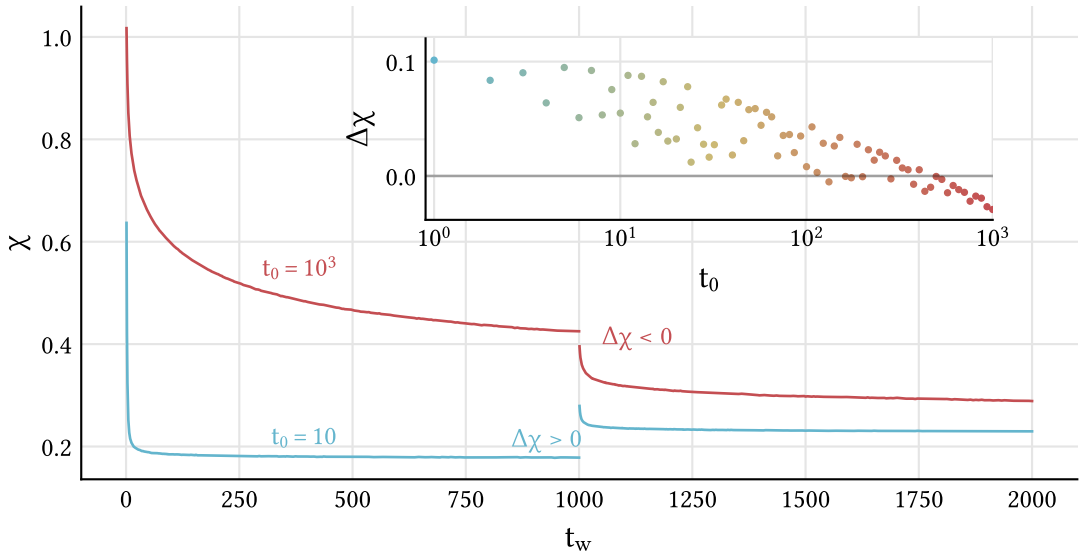


Figura 4.1: Susceptibilidad χ para $T_1 = 0.9, T_2 = 0.5, L = 8$ promediadas sobre 31500 *muestras con valores de $t_0 \in [1, 10^3]$, color en el original. El rejuvenecimiento fuerte desaparece cuando $t \simeq \frac{1}{2}10^3$.*

(figura 4.3). Analizando los resultados, vemos que Δt crece con t_0 y disminuye con t_w^1 y con la diferencia de temperaturas T_1, T_2. Notamos que el comportamiento del retículo $L = 8$ es anómalo respecto al de las redes $L = 48$ y $L = 80$, probablemente por su pequeño tamaño. Para $t_w^1 > 10^6$, éste llega a mostrar un rejuvenecimiento inmediato, alcanzando la curva de referencia en una sola iteración de Montecarlo. Además, los tiempos de recuperación Δt son notablemente más pequeños para el retículo $L = 8$, cinco órdenes de magnitud menores que $L = 48$ y $L = 80$. Esto es achacable a que su evolución es notablemente más rápida.

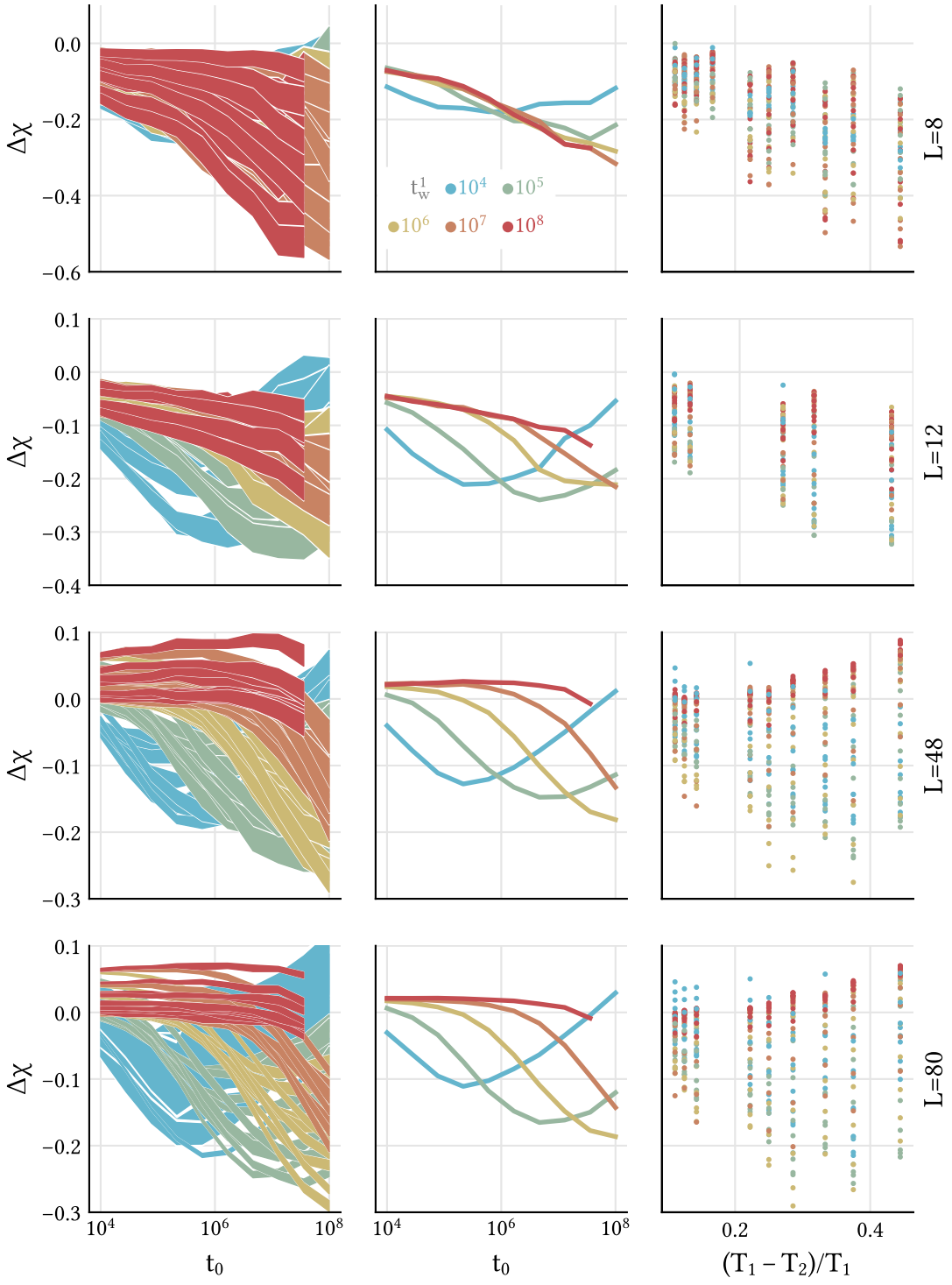


Figura 4.2: En la columna izquierda se muestran bandas con las medidas de $\Delta\chi$ tomadas para cada grupo de parámetros $(t_w^1, t_0, \frac{T_1-T_2}{T_1})$, correspondiendo el grosor a la incertidumbre a 1σ. En la columna central, se ven esas mismas medidas promediando sobre las distintas temperaturas medidas. A la derecha se muestra el valor medio del $\Delta\chi$ según la relación entre las temperaturas T_2 y T_1, de nuevo para todos los grupos de parámetros medidos. En color se indica t_w^1, como muestra la leyenda.

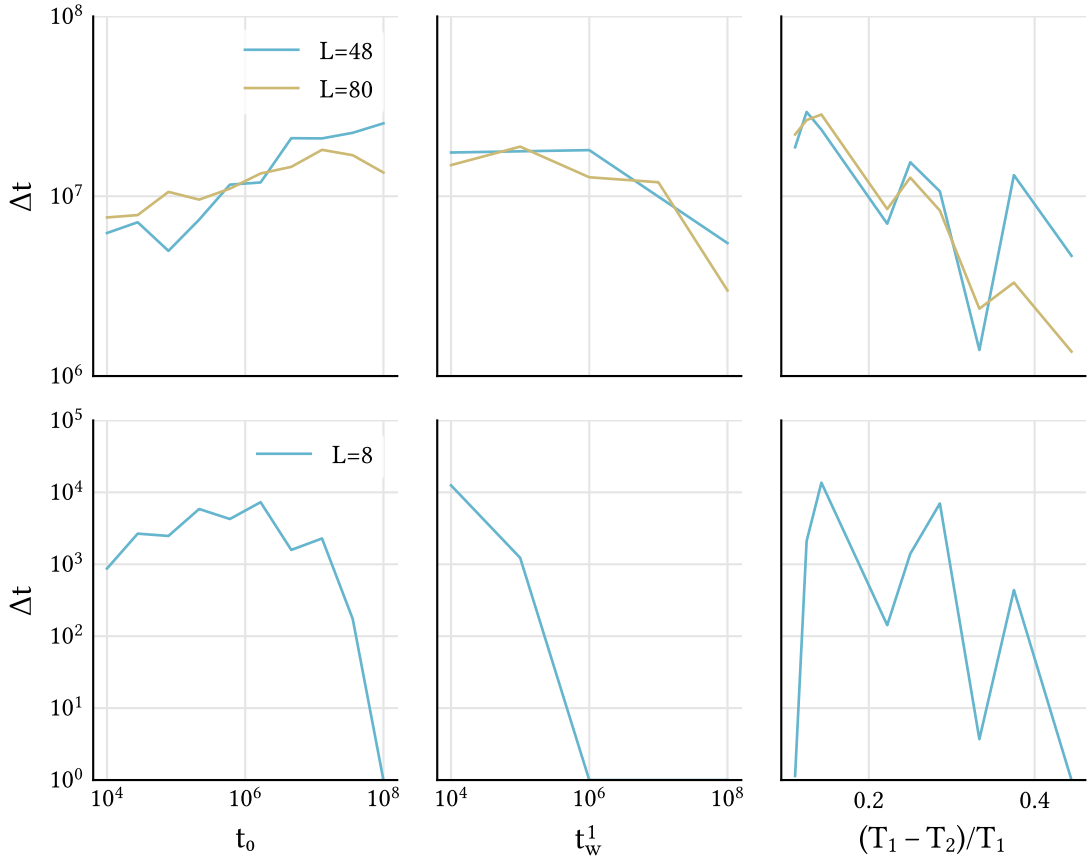


Figura 4.3: Se muestra el tiempo Δt necesario para que el sistema recupere la curva de referencia tras volver a T_1 en función de t_0,t_w^1 o $\frac{T_1-T_2}{T_1}$ y promediando sobre el resto. Se indica en color el tamaño del sistema simulado.

Capítulo 5

Conclusiones

A lo largo del trabajo realizado se ha comprobado que la simulación de vidrios de espín es un tema delicado y duro, requiriendo grandes cantidades de tiempo tanto humano como de CPU. A pesar de la simplicidad del modelo empleado, se ha visto como el comportamiento del sistema era rico y variado.

A la hora de emplear tanto el *cluster* como Janus, se tuvieron diversos problemas técnicos que requirieron adaptar las simulaciones numerosas veces para evitar la pérdida de datos. Las simulaciones de vidrios de espín duran días, aumentando la probabilidad de aparición de errores, especialmente en máquinas con varios años de uso como Janus.

En el estudio del rejuvenecimiento, se ha comprobado como el rejuvenecimiento fuerte era especialmente frágil, mostrando la mayoría de los datos tomados rejuvenecimiento débil. La frecuencia de medición ω de la susceptibilidad, a través del tiempo de correlación t_0, juega un papel importante en el efecto junto al tiempo en que se cambia la temperatura del sistema, t_w^1.

La memoria ha resultado ser un efecto mucho más robusto, aunque su forma funcional no se haya conseguido analizar con la misma precisión que el rejuvenecimiento. En esta magnitud es especialmente claro el comportamiento anómalo de retículos pequeños ($L = 8$), lo que justifica la simulación de sistemas cada vez más grandes y la necesidad de recursos computacionales cada vez más potentes.

Apéndice A

Bootstrapping

En ocasiones se quiere hallar el valor de un observable, pero no se posee una magnitud que sea adecuado asociar con el error de la medida; por ejemplo, la desviación estándar de una lista de valores no es un estimador de la incertidumbre de su media. Esto se solucionó empleando técnicas de remuestreo, concretamente *bootstraping* (remuestreo con reemplazamiento) [12].

Supongamos que tenemos una lista de N valores v_i, y queremos hallar su media. Si bien es trivial emplear $\frac{1}{N}\sum_{i=1}^{N} v_i$ para ello, no conocemos cuál es la incertidumbre de dicha medición.

En un *bootstraping*, se toman de forma aleatoria M muestras con una fracción f de los elementos originales. Si por ejemplo $v = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$ y $f = 1/3$, empleando $M = 5$ obtenemos 5 listas de $9/3 = 3$ elementos. Notar como el muestreo es con reemplazamiento, y por lo tanto puede haber repeticiones. A continuación, calculamos el valor de la función sobre cada muestra, obteniendo M valores para la función, en este caso la media:

$$1, 2, 3, 4, 5, 7, 8, 9 \longrightarrow \begin{cases} 5, 5, 2 \\ 8, 1, 7 \\ 1, 9, 5 \\ 7, 7, 4 \\ 3, 9, 7 \end{cases} \overset{Media}{\longrightarrow} \begin{cases} 4.0 \\ 5.3 \\ 5.0 \\ 6.0 \\ 6.3 \end{cases}$$

Por último, calculamos la media y la desviación estándar de los M valores obtenidos, obteniendo así un estimador de la incertidumbre de la función y su valor medio.

La elección del parámetro M es fácil, ya que basta con que sea lo suficientemente grande como para garantizar que todos los valores se empleen. Como las listas sobre las que se realizó *bootstraping* en este trabajo tenían unos cien elementos, se escogió $M = 10^3$. Como valor para el tamaño de los intervalos se escogió una fracción $f = 1/e$ de los datos, como propone el *Numerical Recipes* [13].

Apéndice B

Números aleatorios

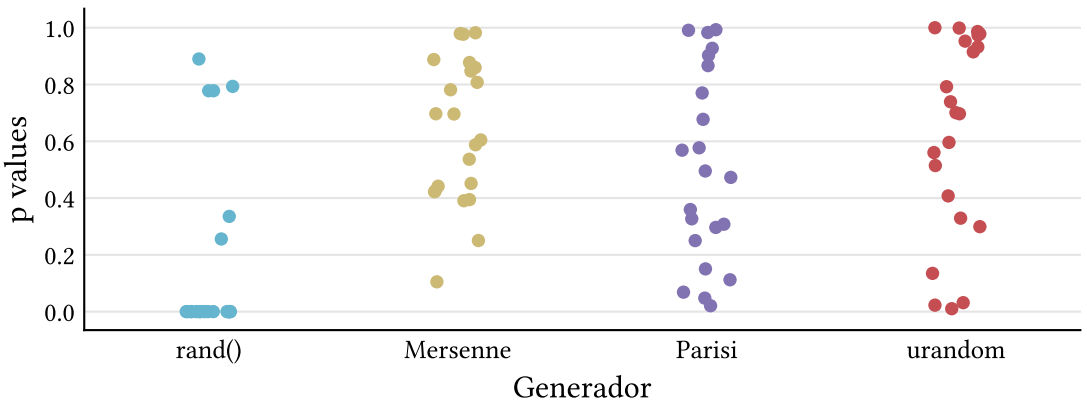
La elección del generador de numeros pseudoaleatorios es crucial para realizar una simulación de Montecarlo, ya que puede sesgar notablemente los resultados [14][15] o convertirse en el cuello de botella del programa si es muy lento.

A la hora de escoger un generador de números pseudoaleatorios, se primó la correctitud en primer lugar y la velocidad en segundo. En la tabla y gráfica siguiente pueden verse algunos de los resultados de la suite de tests `dieharder`. Si suponemos como hipótesis nula, H_0, que el generador de números pseudoaleatorios sea indistinguible de uno realmente aleatorio, podemos realizar diversos tests sobre los números y ver el *p-value* de la distribución obtenida conociendo la esperada bajo números realmente aleatorios. Por ejemplo, si lanzamos una moneda virtual, esperamos ver una distribución binomial, que podemos comparar con la obtenida para hallar el *p-value*.

Los propios *p-values* están distribuídos de forma estadística, así que esperamos ver fallar incluso a un generador perfecto un tanto por ciento determinado de las veces. Esto hace que no sólo halla que rechazar generadores con un *p-value* muy bajo, sino también los que tengan un *p-value* muy alto. La suite de tests, por defecto, es conservadora y anuncia como "débiles" los tests con $p > 0.995$ y $p < 0.05$ y como "fallados" los tests con $p > 0.9995$ y $p < 0.005$. Todos los generadores usados pasan todos los tests realizados (ver figura y tabla a continuación) salvo por el generador por defecto de C compilando con `Glibc`. En la fuente empleada se indica el veredicto de cada prueba: pass, `weak` o *failed*.

En vista de los resultados, se optó por escoger una implementación [16] del algoritmo *Mersenne Twister*, empleando como periodo $2^{601} - 1$, por ser rápido y correcto.

	Glibc `rand()` *4.0·10⁷ rand/s*	Mersenne twister *5.7·10⁷ rand/s*	Parisi-Rapuano *5.4·10⁷ rand/s*	`/dev/urandom` *2.8·10⁷ rand/s*
`diehard birthdays`	0.77798187	0.45147138	0.99298037	0.98661661
`diehard operm5`	0.77805163	0.69607618	0.57700663	0.97395944
`diehard rank 32x32`	*0.00000000*	0.85981358	0.25006856	0.32911572
`diehard rank 6x8`	0.33539556	0.42226451	0.99099839	0.29919422
`diehard bitstream`	*0.00000000*	0.58769831	0.47283337	0.59593074
`diehard opso`	0.25581034	0.25032842	0.32690586	`0.99882142`
`diehard oqso`	0.88991963	0.39452423	0.11213205	0.69691232
`diehard dna`	*0.00000000*	0.10489982	0.56864535	0.91507214
`diehard count 1s str`	*0.00000000*	0.98221540	0.04749207	0.03141170
`diehard count 1s byt`	*0.00000000*	0.60468106	0.02067600	0.02277030
`diehard parking lot`	*0.00000000*	0.78148856	0.35955403	0.40756286
`diehard 2dsphere`	*0.00000000*	0.97711410	0.86656341	0.79222547
`diehard 3dsphere`	*0.00000000*	0.87744534	0.67729253	0.51452466
`diehard squeeze`	*0.00000000*	0.88793557	0.92759964	0.73919529
`diehard sums`	*0.00000000*	0.84744756	0.90264575	0.56030728
`diehard runs`	0.79338389	0.69696354	0.06846611	`0.99999754`
`diehard craps`	*0.00000000*	0.39061750	0.98312026	0.93221284
`marsaglia tsang gcd`	*0.00000000*	0.80756474	0.15047973	0.95279822
`sts monobit`	*0.00000000*	0.97927930	0.49533581	0.97838676
`sts runs`	*0.00000000*	0.44167854	0.77021754	0.70077385
`sts serial`	*0.00000000*	0.53670356	0.30774783	0.01020360
`rgb bitdist`	*0.00000000*	0.42450528	0.29655434	0.13467538



Apéndice C

Multispin coding

La simulación de vidrios de espín requiere un gran número de réplicas, necesitándose correr varias simulaciones en paralelo. La naturaleza binaria de los espines, que sólo necesitan un bit para ser codificados, hace que sea conveniente "empaquetar" varios en las palabras de 32 o 64 bit de un ordenador en las iteraciones del algoritmo de Metrópolis-Hasting [6] empleado, obteniendo paralelismo a nivel de bits como se verá a continuación.

Supongamos, para simplificar la explicación, un modelo de Ising simple con hamiltoniano $\mathscr{H} = -\sum_{\langle ij \rangle} s_i s_j$. En cada iteración del algoritmo de Metrópolis-Hasting, necesitamos hallar la energía de los espines y si es necesario voltearlos. Sin optimizaciones, el algoritmo para actualizar cada espín sería el siguiente:

1. Calcular la energía con los 6 vecinos. Esto requiere una suma de 6 términos $\sum_j s_j$, una multiplicación $s_i \cdot \sum_j s_j$ y una negación por el signo menos del hamiltoniano.

2. Calcular la probabilidad $e^{-\beta \mathscr{H}_i}$ de volteo. La forma más eficiente de ejecutar este paso es construir una *look up table* con los términos precalculados para los posibles valores de la energía $\mathscr{H}_i \in \{0, \pm 4, \pm 8, \pm 12\}$, lo que además nos ahorra la negación aritmética anterior sin más que ordenar la tabla al revés.

3. Voltear el espín, si procede. Esto requiere una negación aritmética y el cálculo de un número aleatorio.

Notar que en el algoritmo se han despreciado los accesos a la memoria cache[1].

La primera optimización que se puede realizar consiste en explotar el carácter binario de los espines y emplear únicamente operaciones lógicas, mucho más rápidas que las aritméticas. Para ello, pasamos de variables $s \in \{\pm 1\}$ a variables $\sigma \in \{0, 1\}$. Vemos que $s = 2\sigma + 1$, por lo que podemos traducir las sumas $\sum_i s_i$ a sumas en σ con facilidad. Además, las multiplicaciones se pueden sustituir por un XOR y una negación asignando $+1 \longrightarrow 1$ y $-1 \longrightarrow 0$:

[1] Incluso para $L = 40$, los datos necesarios (~L^3 palabras de 64 bits para el código empleado) caben cómodamente en la memoria cache de Memento, el ordenador empleado.

s_1	s_2	$s_1 \cdot s_2$
+1	+1	+1
+1	-1	-1
-1	+1	-1
-1	-1	+1

σ_1	σ_2	$\neg(\sigma_1 \oplus \sigma_2)$
1	1	1
1	0	0
0	1	0
0	0	1

Una vez que se ha pasado de operaciones aritméticas a lógicas se puede explotar paralelismo en las propias palabras binarias empleadas, en este caso de 64 bits. En cada palabra cabrían 64 espines, pero como la suma $\sum_j \sigma_j$ puede valer como máximo 6 (`110` en binario) necesitamos 3 bits por espín, de forma que caben 21 espines de diferentes sistemas por palabra binaria[2]:

Representación	
Espines	↑↑↓↑↓↑↓↓↓↓↓↑↓↑↑↓↑↓↑↑
Octal	`1101010000000101101011`
Hexadecimal	`1208200001048209`
Binario	`0.001.001.000.001.000.001.000`⋯

Notar como al haber un espín por cada tres posiciones las posibles palabras binarias en octal sólo pueden tener los números 0 (000) o 1 (001); cualquier otra alternativa implicaría más de un espín por posición. Por este motivo, la representación octal de las palabras corresponde directamente a los espines, haciéndola especialmente útil en la fase de *debugging*.

El algoritmo mencionado previamente pasa a ser el siguiente:

1. Calculamos la energía con los vecinos. Para ello, comenzamos por hacer la suma $\sum_j \sigma_j$ de los seis primeros vecinos del espín σ_i:

```
  0 000 001 000 000 001 001 000 000 000 000
  0 001 000 001 000 000 001 001 001 000 000
  0 001 001 000 001 001 000 000 001 000 001
+ 0 001 001 001 001 000 000 001 001 000 000
  0 000 000 000 000 001 001 000 001 000 001
  0 000 000 001 001 000 001 001 001 000 001
-----------------------------------------------
  0 011 011 011 011 011 100 011 101 000 011
```

   donde se han truncado las palabras binarias a 31 bits por motivos de espacio. A continuación, multiplicamos por el σ_i de $\mathscr{H}_i = \sigma_i \sum \sigma_j$, donde $\sum \sigma_j$ es la cantidad que acabamos de calcular. Para ello, hacemos un XOR lógico; no es necesario negarlo, ya que basta con alterar el orden de la *look up table* en la que se guardan las probabilidades de la exponencial $e^{\beta \mathscr{H}_i}$.

```
  0 011 011 011 011 011 100 011 101 000 011
⊕ 0 001 001 000 001 001 000 001 001 000 000
-----------------------------------------------
  0 010 010 011 010 010 100 010 100 000 011
```

2. Consultamos *para cada uno de los 21 grupos de tres espines* el índice correspondiente en la *look up table*, obteniendo las 21 probabilidades $e^{-\beta \mathscr{H}}$. Con ellas, fabricamos una nueva palabra binaria de 64 bits que contiene 21 grupos de tres bits. Si el espín correspondiente se

[2] El bit sobrante ($21 \times 3 = 63 < 64$) se ignora en el tratamiento y en el software.

va a girar, se coloca 111, si no `000`. Basta con hacer un XOR de σ_i con esta palabra binaria para girar los correspondientes espines.

Hay que notar que el nuevo algoritmo necesita un bucle sobre los 21 espines de cada palabra, haciendo que a pesar de realizar operaciones más simples (binarias frente a aritméticas) sea más lento. No obstante, requiere 21 veces menos números aleatorios, y calcula en paralelo la evolución de 21 sistemas, que en el caso de este trabajo serán 21 muestras diferentes.

En los *benchmarks* realizados se comprobó que este algoritmo era tres veces más lento[3], pero el gran aumento en estadística que se tiene lo hace imprescindible: para sistemas de $L < 20$, son necesarias miles de muestras para obtener los valores medios de algunos observables, como la correlación. Como referencia, la simulación en un PC (Intel Core i7 cuarta generación) necesita aproximadamente 20 ns por espín sin *multispin coding* y 60 ns por cada grupo de 21 espines con él.

Cuando se simulan sistemas con J variable hay que modificar ligeramente el algoritmo [17]: en lugar de poder sacar los acoplos J_{ij} de la suma $\sum J_{ij}\sigma_i\sigma_j$ como una constante, es necesario multiplicarlos primero por los σ_j. Para cada espín σ_i,

$$\sum_j J_{ij}\sigma_i\sigma_j = \sigma_i \sum_j J_{ij}\sigma_j \tag{C.1}$$

De forma que hay que realizar seis multiplicaciones $J_{ij}\sigma_j$ antes de sumar los espines de los seis primeros vecinos y multiplicar por σ_i. En el siguiente apéndice se muestra un ejemplo de implementación.

[3] Sin optimizaciones agresivas en el compilador (`-ffast-math -O2` en `clang` y `gcc`), es dos veces más lento.

Apéndice D

Código fuente

El código fuente de la memoria, incluyendo tanto la librería desarrollada para las simulaciones como utilidades para realizarlas, junto al código LaTeXde la memoria y el empleado para generar las gráficas, está disponible en un repositorio público de `Github` en lugar de en un anexo por ser relativamente extenso:

`github.com/redpointyjackson/tfg`

El código para las simulaciones se ha escrito en C99, mientras que el análisis de los datos se ha realizado en Julia [5] y Python.

A continuación se reproducen las dos funciones principales de la librería principal, el cálculo de $\mathscr{H}_i = -\sum_j J_{ij}\sigma_i\sigma_j$ y el algoritmo de Metropolis-Hasting [6], en las que se ve el uso de operaciones binarias para paralelizar la simulación mediante *multispin coding* (Apéndice C). Los L^3 espines se guardan en un vector unidimensional de palabras binarias de 64 bits, y se emplean tablas precalculadas para calcular la posición de los primeros vecinos (`guide_left` y similares) y las probabilidades $e^{\beta\mathscr{H}_i}$ (el campo `probs` dentro de la estructura `SG`).

```
uint64_t local_energy(struct net* SG, int64_t idx, int64_t x, int64_t y, int64_t z){
  uint64_t* S     = SG->spins;
  uint64_t* Jr    = SG->J_right;
  uint64_t* Ju    = SG->J_up;
  uint64_t* Jf    = SG->J_front;

  uint64_t right  = S[idx] ^ Jr[ idx                  ] ^ S[ idx + guide_right[x]  ];
  uint64_t left   = S[idx] ^ Jr[ idx + guide_left[x]   ] ^ S[ idx + guide_left[x]   ];
  uint64_t up     = S[idx] ^ Ju[ idx                  ] ^ S[ idx + guide_up[y]     ];
  uint64_t down   = S[idx] ^ Ju[ idx + guide_down[y]   ] ^ S[ idx + guide_down[y]   ];
  uint64_t front  = S[idx] ^ Jf[ idx                  ] ^ S[ idx + guide_front[z]  ];
  uint64_t behind = S[idx] ^ Jf[ idx + guide_behind[z] ] ^ S[ idx + guide_behind[z] ];

  return right + left + up + down + front + behind;
}
```

```
void metropolis(struct net* SG){
  int64_t curr_idx = 0;
  int64_t L = SG->L;

  for(int64_t z=0;z<L;z++){
    for(int64_t y=0;y<L;y++){
      for(int64_t x=0;x<L;x++){

        uint64_t E = local_energy(SG, curr_idx, x, y, z);
        double randomnum = RANDOM_F;

        for(int_fast8_t spinidx=0; spinidx<21; spinidx++){

          uint64_t Pidx = E;
          Pidx = Pidx >> 3*spinidx;
          Pidx = Pidx &  0x7; // Select the first 3 bits.

          double P = SG->probs[Pidx];

          if (Pidx <= 3 || P > randomnum){
            uint64_t selectmask = 0x1;
            selectmask = selectmask << 3*spinidx;
            SG->spins[curr_idx] ^= selectmask; // Flip current .
          }
        }
        curr_idx++;
      }
    }
  }
  SG->mc_steps++;
}
```

Bibliografía

[1] K. Jonason, E. Vincent, J. Hammann, J. P. Bouchaud, and P. Nordblad. Memory and chaos effects in spin glasses. *Phys. Rev. Lett.*, 81:3243–3246, Oct 1998.

[2] S. Miyashita and E. Vincent. A microscopic mechanism for rejuvenation and memory effects in spin glasses. *The European Physical Journal B - Condensed Matter and Complex Systems*, 22(2):203–211, 2001.

[3] V. Cannella and J. A. Mydosh. Magnetic ordering in gold-iron alloys. *Phys. Rev. B*, 6:4220–4237, Dec 1972.

[4] S. F. Edwards and P. W. Anderson. Theory of spin glasses. *Journal of Physics F: Metal Physics*, 5(5):965, 1975.

[5] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B. Shah. Julia: A fresh approach to numerical computing. *SIAM Review*, 59(1):65–98, 2017.

[6] Christian P. Robert. The metropolis-hastings algorithm, 2015.

[7] J. A. Mydosh. *Spin glasses: an experimental introduction*. Taylor and Francis, 1993.

[8] F. Belletti, M. Cotallo, A. Cruz, L. A. Fernández, A. Gordillo, A. Maiorano, F. Mantovani, E. Marinari, V. Martín-Mayor, A. Muñoz-Sudupe, D. Navarro, S. Pérez-Gaviro, J. J. Ruiz-Lorenzo, S. F. Schifano, D. Sciretti, A. Tarancón, R. Tripiccione, and J. L. Velasco. Simulating spin systems on ianus, an fpga-based computer. *Computer Physics Communications*, 178(3):208–216, 2 2008.

[9] Laurence Jacobs and Claudio Rebbi. Multi-spin coding: A very efficient technique for Montecarlo simulations of spin systems. *Journal of Computational Physics*, 41(1):203 – 210, 1981.

[10] Enzo Marinari, Giorgio Parisi, and Juan J Ruiz-Lorenzo. Low T dynamical properties of spin glasses smoothly extrapolate to T = 0. *Journal of Physics A: Mathematical and General*, 35(32):6805, 2002.

[11] F. Belletti, A. Cruz, L. A. Fernandez, A. Gordillo-Guerrero, M. Guidetti, A. Maiorano, F. Mantovani, E. Marinari, V. Martin-Mayor, J. Monforte, A. Muñoz Sudupe, D. Navarro, G. Parisi, S. Perez-Gaviro, J. J. Ruiz-Lorenzo, S. F. Schifano, D. Sciretti, A. Tarancon, R. Tripiccione,

and D. Yllanes. An in-depth view of the microscopic dynamics of Ising spin glasses at fixed temperature. *Journal of Statistical Physics*, 135(5):1121–1158, 2009.

[12] B. Efron. Bootstrap methods: Another look at the jackknife. *The Annals of Statistics*, 7(1):1–26, 1979.

[13] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical recipes in C: the art of scientific computing*. Cambridge University Press, second edition edition, 2002. Ver *Quick-and-dirty Monte Carlo: The bootstrap method*.

[14] Alan M Ferrenberg, DP Landau, and Y Joanna Wong. Monte carlo simulations: Hidden errors from ''good''random number generators. *Physical Review Letters*, 69(23):3382, 1992.

[15] Heiko Bauke and Stephan Mertens. Pseudo random coins show more heads than tails. *Journal of Statistical Physics*, 114(3):1149–1169, 2004.

[16] Mutsuo Saito and Makoto Matsumoto. SIMD oriented Fast Mersenne Twister (SFMT). `http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/SFMT/index.html`, 2017. Último acceso a la página web en el 2017.

[17] Ye Fang, Sheng Feng, Ka-Ming Tam, Zhifeng Yun, Juana Moreno, J. Ramanujam, and Mark Jarrell. Parallel tempering simulation of the three-dimensional Edwards–Anderson model with compact asynchronous multispin coding on GPU. *Computer Physics Communications*, 185(10):2467 – 2478, 2014. La sección 4.3 es la más relevante.

Silueta de la portada de *Freepik* (`www.flaticon.com`).