Trabajo de Fin de Grado de Física

Cálculo numérico en teoría cuántica de campos de la materia condensada

Autor:
Jorge Alda Gallo

Directores:
David Zueco Láinez
Isaac Fernando Quijandría Díaz

Departamento de Física de la Materia Condensada
Facultad de Ciencias, Universidad de Zaragoza

Junio de 2015

"El científico no estudia la naturaleza por la utilidad que le pueda reportar;
la estudia por el gozo que le proporciona,
y este gozo se debe a la belleza que hay en ella"
Henri Poincaré

Me gustaría agradecer a mi familia por su infinita paciencia y comprensión conmigo, y por su constante impulso y apoyo para seguir adelante y perseguir nuevas metas. También a mis amigos y compañeros por no fallarme nunca en los momentos complicados y hacerlos siempre un poco más fáciles.

Por supuesto, no me puedo olvidar de David y Fernando. Me han abierto la puerta a un mundo fascinante repleto de nuevos retos, y con su tutela he podido superarlos con éxito. Ellos han convertido este trabajo en toda una experiencia vital.

Para la elaboración de este trabajo he sido beneficiario de la beca JAE Intro 2014 para la introducción a la investigación concedida por el CSIC.

Índice general

Introducción **1**

Objetivos **2**

1 Conceptos de mecánica cuántica **3**
1.1. Operador de evolución . . . 3
1.2. Matriz densidad . . . 4
1.3. Entrelazamiento . . . 5
1.4. Espacio de Fock . . . 6
1.5. Teoría cuántica de campos . . . 7
1.6. El principio variacional . . . 7

2 Estados producto de matrices **8**
2.1. Tensores . . . 8
2.2. Estados cuánticos como tensores . . . 9
2.3. Utilidad de los estados tensoriales . . . 10

3 Estados producto de matrices continuos **11**
3.1. Paso al continuo . . . 11
3.2. Expresiones para algunas magnitudes . . . 12
3.3. Dinámica disipativa en el sistema auxiliar . . . 12
3.4. Extensión a campos acoplados . . . 14

4 Metodología **16**
4.1. Algoritmo para un solo campo . . . 16
4.2. Algoritmo para dos campos acoplados . . . 18

5 Modelos y resultados **20**
5.1. Modelo de Lieb-Liniger . . . 20
5.2. Dos campos acoplados . . . 23

Conclusiones **27**

Bibliografía **28**

A Cálculos con cMPS **30**
A.1. Normalización . . . 31
A.2. Funciones de correlación campo-campo . . . 32
A.3. Funciones de correlación densidad-densidad . . . 33

B Código de los programas **34**
B.1. Modelo Lieb-Liniger en un campo . . . 34
B.2. Cálculo de funciones de correlación . . . 43

Introducción

La teoría cuántica de campos es el lenguaje natural para la descripción de la física de la materia condensada. Superfluidez, superconductividad, condensación de Bose-Einstein o líquidos de Fermi son solo algunos de los fenómenos cuya comprensión requiere el uso de campos cuánticos [1]. El caso de los líquidos de Fermi es paradigmático del éxito de la teoría de perturbaciones. Sin embargo, en los sistemas unidimensionales las interacciones se ven reforzadas [2], por lo que la teoría de perturbaciones no es aplicable.

Para ir más allá del régimen perturbativo, se puede emplear la discretización de los campos y su simulación computacional. La existencia de una distancia mínima conlleva el problema de la introducción de un máximo en las energías y momentos. Para elevar este corte (*cutoff*) hasta energías suficientemente altas como para que no tengan relevancia física, hay que disminuir consiguientemente la discretización. Los niveles de discretización necesarios hacen completamente inviables las perspectivas de una simulación.

Como método alternativo, en el año 2010 Verstraete y Cirac [3] propusieron un estado de prueba (*ansatz*) para el método variacional, el **estado de producto de matrices continuo**. Esta solución tiene la ventaja de que necesita muy pocos parámetros para describir los estados físicamente relevante, lo cual es una clara ventaja a la hora de realizar simulaciones. Además, otro punto a favor es que su formulación matemática tiene una interpretación física directa, como un sistema auxiliar discreto interaccionando con el campo.

El estado de producto de matrices continuo puede ser generalizado para el caso de varios campos que interaccionan entre sí [4]. Como se verá, esto permite la reproducción de una transición de fase fuera del régimen perturbativo con un esfuerzo computacional muy moderado.

La organización de esta memoria es como sigue: tras una exposición de los objetivos del trabajo de fin de grado, se procederá a la revisión de los conceptos de mecánica cuántica estudiados durante el grado necesarios en la sección 1. En la sección 2 se presentan los estados producto de matrices, el predecesor para casos discretos de los estados producto de matrices, y con cuyo estudio se comprenderán las ventajas que ofrecen para la descripción de sistemas físicos. En la sección 3 se describen los estados producto de matrices continuos y su generalización para campos acoplados. Una vez conocida toda la teoría, en la sección 4 se explican los algoritmos utilizados para las simulaciones, destacando los problemas encontrados durante su ejecución y las soluciones adoptadas. Los modelos empleados y los resultados obtenidos mediante las simulaciones se presentan en la sección 5. Finalmente se exponen las conclusiones alcanzadas en la realización del trabajo y las posibles aplicaciones que ofrece en un futuro.

Objetivos

El primer objetivo que debe cumplir el presente trabajo es la comprensión y asimilación de los fundamentos teóricos en los que se basan las simulaciones que se van a realizar:

- Identificación del entrelazamiento cuántico como la magnitud adecuada para caracterizar los estados físicos.
- Revisión bibliográfica de los métodos de simulación basados en el entrelazamiento, las llamadas redes de tensores, haciendo especial énfasis en los estados producto de matrices.
- Estudio de la generalización de estos métodos a campos mediante los estados producto de matrices continuos. Cálculos, interpretación física.
- Ampliación del formalismo a campos que interaccionan.

Una vez que estén fijados los conceptos esenciales, el objetivo es aprovecharlos para diseñar y ejecutar simulaciones numéricas con las que obtener nuevos resultados:

- Elaboración de un programa para simular un campo cuántico partiendo de los cálculos vistos en la teoría.
- Comprobación del código comparando los resultados obtenidos con métodos analíticos.
- Extensión a dos campos en interacción.
- Localización y caracterización de una transición de fase.

1 Conceptos de mecánica cuántica

1.1. Operador de evolución

Uno de los postulados de la mecánica cuántica (el quinto para [5], el segundo para [6]) establece que las funciones de onda evolucionan en el tiempo según la ecuación de Schrödinger. Esto implica que las funciones de onda en dos tiempos distintos t_0 y t_1 están relacionadas mediante una transformación U, que debe ser unitaria para que se conserve la densidad de probabilidad:

$$\psi(t_1) = U(t_1, t_0)\psi(t_0) \qquad U(t_1, t_0)U^\dagger(t_1, t_0) = \mathbb{1} \tag{1.1}$$

Además, el operador evolución temporal U debe respetar las operaciones de grupo, ya que la evolución desde un tiempo t_0 hasta t_1 seguida de la evolución desde t_1 a t_2 debe ser equivalente a la evolución desde t_0 hasta t_2.

$$U(t_2, t_0) = U(t_2, t_1)U(t_1, t_0) \tag{1.2}$$

En concreto, si se realiza una evolución en un tiempo infinitesimal, estará dada por el generador de la transformación, el hamiltoniano H. Esto significa que el operador U debe cumplir la ecuación diferencial (1.3). La misma ecuación se puede obtener introduciendo (1.1) en la ecuación de Schrödinger:

$$\partial_t U = -iHU \qquad \partial_t U^\dagger = iU^\dagger H \tag{1.3}$$

La solución general a dicha ecuación diferencial es la siguiente [7]:

$$U(s, s_0) = \mathcal{T} \exp \int_{s_0}^{s} -iH(t)\mathrm{d}t \equiv \sum_{n=0}^{\infty} \frac{(-i)^n}{n!} \int_{s_0}^{s} \cdots \int_{s_0}^{s} \mathcal{T} H(t_1) \cdots H(t_n)\mathrm{d}t_1 \cdots \mathrm{d}t_n \tag{1.4}$$

donde $\mathcal{T}$ es el operador de ordenación temporal:

$$\mathcal{T} A(t)B(t') = \begin{cases} A(t)B(t') & \text{si} \quad t > t' \\ B(t')A(t) & \text{si} \quad t' > t \end{cases} \tag{1.5}$$

La necesidad de la ordenación de los operadores es una consecuencia de que el hamiltoniano a diferentes tiempos no tiene por qué conmutar, y el orden lo establece la ecuación 1.3 (por ejemplo, para el operador $U^\dagger$ habría que emplear el orden opuesto).

Todo esto se puede generalizar a una evolución no unitaria definida por la siguiente ecuación:

$$\partial_s U = UH \tag{1.6}$$

La solución a esta ecuación se expresa de nuevo como el desarrollo en serie de la exponencial del operador hamiltoniano:

$$U(s_0, s) = \mathcal{P} \exp \int_{s_0}^{s} H(t)\mathrm{d}t \equiv \sum_{n=0}^{\infty} \frac{1}{n!} \int_{s_0}^{s} \cdots \int_{s_0}^{s} \mathcal{P} H(t_1) \cdots H(t_n)\mathrm{d}t_1 \cdots \mathrm{d}t_n \tag{1.7}$$

El operador $\mathcal{P}$ realiza el orden según el parámetro s. De la ecuación (1.6) se sigue que será el siguiente:

$$\mathcal{P} A(s)B(s') = \begin{cases} A(s)B(s') & \text{si} \quad s < s' \\ B(s')A(s) & \text{si} \quad s' < s \end{cases} \tag{1.8}$$

Emplearemos la imagen de interacción (o de Dirac), que se basa en la separación del hamiltoniano en dos partes, $H = H_0 + V$. H_0 representa la evolución libre y V un término de interacción. El operador evolución en la imagen de interacción se define como

$$U_I(s_0, s) = U(s_0, s)\mathrm{e}^{-H_0(s-s_0)} \tag{1.9}$$

Asumiendo invarianza bajo traslaciones, $U(s_0, s) = U(0, s-s_0)$, podemos escribir la expresión anterior como:

$$U_I(0,x) = U(0,x)\mathrm{e}^{-H_0 x} \tag{1.10}$$

Derivando según (1.6):

$$\partial_x U_I = U(0,x)H\mathrm{e}^{-H_0 x} - U(0,x)H_0\mathrm{e}^{-H_0 x} = U_I(0,x)V \tag{1.11}$$

Integrando de nuevo (con la condición de contorno de $U_I(0,0) = \mathbb{1}$) y operando, podemos reescribir el operador evolución:

$$U_I(0,x) = U(0,x)\mathrm{e}^{-H_0 x} = \mathbb{1} + \int_0^x \mathrm{d}s U(0,s)V\mathrm{e}^{-H_0 s} \tag{1.12}$$

$$U(0,x) = \mathrm{e}^{H_0 x} + \int_0^x \mathrm{d}s U(0,s)V\mathrm{e}^{H_0(x-s)} \tag{1.13}$$

La solución a esta ecuación integral se puede obtener mediante el método iterativo. Para ello emplearemos una solución aproximada $U^{(n)}$ y la insertaremos en la expresión (1.13), para obtener la siguiente aproximación $U^{(n+1)}$. La solución exacta se obtiene en el límite $n \to \infty$. Como primera solución, para que sea consistente con el límite en $x = 0$, se toma $U^{(0)} = \mathrm{e}^{H_0 x}$:

$$\begin{aligned} U^{(1)}(0,x) &= \mathrm{e}^{H_0 x} + \int_0^x \mathrm{d}s U^{(0)}(0,s)V\mathrm{e}^{H_0(x-s)} = \mathrm{e}^{H_0 x} + \int_0^x \mathrm{d}s \mathrm{e}^{H_0 x}V\mathrm{e}^{H_0(x-s)} \\ U^{(2)}(0,x) &= \mathrm{e}^{H_0 x} + \int_0^x \mathrm{d}s \mathrm{e}^{H_0 x}V\mathrm{e}^{H_0(x-s)} + \int_0^x \mathrm{d}s' \int_0^{s'} \mathrm{d}s \mathrm{e}^{H_0 s'}V\mathrm{e}^{H_0(s-s')}V\mathrm{e}^{H_0(x-s)} \end{aligned} \tag{1.14}$$

Repitiendo el procedimiento se llega a una expresión en serie para el operador evolución, conocida como **serie de Dyson**:

$$U(0,x) = \sum_{n=0}^{\infty} \int_0^x \mathrm{d}x_n \cdots \int_0^{x_2} \mathrm{d}x_1 \mathrm{e}^{H_0 x_1}V\mathrm{e}^{H_0(x_2-x_1)}V\mathrm{e}^{H_0(x_3-x_2)}\cdots\mathrm{e}^{H_0(x_n-x_{n-1})}V\mathrm{e}^{H_0(x-x_n)} \tag{1.15}$$

1.2. Matriz densidad

Además de la formulación en términos de la función de onda existe otra formulación más general, debida a von Neumann, empleando la *matriz densidad*. Este enfoque tiene la ventaja de poder tratar conjuntos estadísticos de estados además de estados puros [5, Complemento CIII]. Si hay una probabilidad (clásica) p_i de que el estado del sistema sea $|\psi_i\rangle$, la matriz densidad se define como

$$\rho = \sum_i p_i |\psi_i\rangle \langle\psi_i| \tag{1.16}$$

La matriz densidad es un operador hermítico, definido positivo y de traza igual a uno.

Si todos los p_i son nulos a excepción de uno, entonces el estado es puro, y se puede describir mediante una función de ondas; en caso contrario es un estado mezcla. Hay distintas pruebas para comprobar si una matriz densidad ρ corresponde a un estado puro o mezcla. Una de ellas es la entropía de von Neumann (definida de manera análoga a la entropía de Gibbs o de Shanon), que es nula para un estado puro y positiva para un estado mezcla:

$$S(\rho) = -\sum_i p_i \log p_i = -\mathrm{tr}(\rho \log \rho) \tag{1.17}$$

La evolución temporal de la matriz densidad, calculada a partir de la ecuación de Schrödinger, está dada por la ecuación de Liouville-von Neumann:

$$i\hbar\frac{\partial}{\partial t}\rho = [H, \rho] \tag{1.18}$$

El valor esperado de un operador A se calcula según

$$\langle A\rangle = \mathrm{tr}(\rho A) \tag{1.19}$$

Matriz densidad reducida

Podría pensarse que la matriz densidad no aporta nada nuevo con respecto a la función de ondas. Al fin y al cabo, en principio se podría conocer la función de ondas del universo y resolver con ella la ecuación de Schrödinger, y cualquier problema estaría automáticamente resuelto. Pero la matriz densidad proporciona la herramienta necesaria para tratar los componentes de un sistema de forma individual: la matriz densidad reducida.

Si un sistema físico está descrito por la matriz densidad ρ y se puede dividir en dos partes A y B, cuyos espacios de Hilbert admitan las bases $\{|\varphi_i\rangle\}$ y $\{|\chi_j\rangle\}$ respectivamente, a cada uno de los subsistemas le corresponde una matriz densidad reducida obtenida al hacer la traza parcial sobre los grados de libertad de la otra parte:

$$\rho_A = \mathrm{tr}_B\rho = \sum_j \langle\chi_j|\,\rho\,|\chi_j\rangle \tag{1.20a}$$

$$\rho_B = \mathrm{tr}_A\rho = \sum_i \langle\varphi_i|\,\rho\,|\varphi_i\rangle \tag{1.20b}$$

La evolución temporal de cada una de las matrices reducidas sigue obedeciendo la ecuación de Liouville-von Neumann(1.18), y los valores esperados de observables que afecten solo a una de las partes se pueden calcular con (1.19).

1.3. Entrelazamiento

En sistemas de muchas partículas, el espacio de Hilbert es el resultado de realizar el producto tensorial de los espacios de las partículas individuales.

$$\mathcal{H}_T = \mathcal{H}_1 \otimes \mathcal{H}_2 \otimes \cdots \otimes \mathcal{H}_n \tag{1.21}$$

Al hacer esto, aparecen estados que no se pueden separar como el producto tensorial de los estados independientes, lo que se conoce como entrelazamiento. Se define un estado entrelazado como un estado del espacio de Hilbert $\mathcal{H}_T$ que no se puede expresar como un producto tensorial de estados en los diferentes espacios $\mathcal{H}_i$. El ejemplo más típico es el de dos objetos con dos niveles que forman un estado de Bell:

$$|\varphi\rangle = \frac{1}{\sqrt{2}}(|0\rangle\otimes|1\rangle - |1\rangle\otimes|0\rangle) \neq |a\rangle\otimes|b\rangle \tag{1.22}$$

El hecho de que dos objetos estén entrelazados significa que no son independientes entre sí, sino que entre ellos existen correlaciones. En el ejemplo anterior, si se realiza una medición sobre el primer estado y se obtiene el valor $|0\rangle$, automáticamente se sabe que el segundo estará en el estado $|1\rangle$ (o viceversa). El significado del entrelazamiento es que, aunque se tenga un conocimiento completo de un sistema cuántico, eso no implica que se tenga un conocimiento completo de cada una de sus partes por separado.

Entropía de entrelazamiento

En un sistema puro, la condición necesaria y suficiente para que dos de sus partes estén entrelazadas es que sus matrices reducidas correspondan a estados mezcla. Esto permite elaborar diferentes medidas del entrelazamiento, como la entropía de entrelazamiento: En un sistema bipartito, la entropía de entrelazamiento es la entropía de von Neumann de las matrices reducidas. Si el estado está entrelazado, la entropía de entrelazamiento es mayor que cero. Además, se cumple que la entropía de entrelazamiento de ambas partes es igual.

Es bien conocido que la entropía clásica aumenta con el volumen de los sistemas. Cabe preguntarse si sucede lo mismo con la entropía de entrelazamiento. Si se escoge un estado arbitrario [8], la respuesta es que sí. Pero no es cierto en algunos casos concretos, con una gran relevancia física.

En concreto, si el hamiltoniano incluye únicamente interacciones locales y existe una diferencia de energía (*gap*) entre el nivel fundamental y el primer excitado, entonces en el nivel fundamental la entropía de entrelazamiento depende del tamaño de la frontera entre los dos componentes, y por lo tanto están menos entrelazados que un estado general. Esta propiedad se conoce como **ley del área**, y se ilustra en la figura 1.1. Nos restringiremos a sistemas unidimensionales, por lo que la frontera de cualquier bloque está formada por dos sitios, y la entropía estará acotada por una constante.

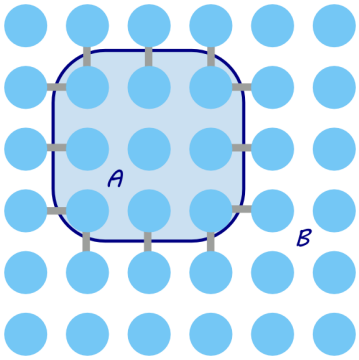

Figura 1.1: En sistemas con interacciones locales y *gaps* de energía, la entropía de entrelazamiento depende del número de enlaces entre las dos partes A y B.

1.4. Espacio de Fock

Hay muchas situaciones físicas en las que el número de excitaciones o de partículas no se mantiene constante. Por ello, hay que emplear un espacio de estados adaptado a un número arbitrario de partículas: el espacio de Fock. Si $\mathcal{H}$ es el espacio de Hilbert correspondiente a una única partícula, el espacio de Fock incluye $\mathbb{C}$ (el vacío, representado con el estado $|\Omega\rangle$), $\mathcal{H}$, $\mathcal{H}^2$, $\mathcal{H}^3$ ..., así como las combinaciones lineales de elementos de los distintos espacios.

La forma conveniente de expresar estos estados, incluyendo la condición de simetrización, es mediante estados de Fock, donde se expresa la ocupación de cada uno de los estados. Estos estados, con un número definido de partículas, constituyen una base del espacio de Fock. Si $a^\dagger(x_n)$ es el operador de creación de una partícula en la posición x_n, un estado de Fock es de la forma

$$|x_1, \cdots x_n\rangle = \frac{1}{\sqrt{n!}} a^\dagger(x_1) \dots a^\dagger(x_n) \, |\Omega\rangle$$

Como estos estados constituyen una base, cualquier otro estado se puede expresar como

combinación lineal de ellos:

$$|\varphi\rangle = \sum_{n=0}^{\infty} \int \mathrm{d}x_1 \cdots \mathrm{d}x_n \phi(x_1, \cdots x_n) a^\dagger(x_1) \cdots a^\dagger(x_n) \, |\Omega\rangle \tag{1.23}$$

Los coeficientes de la combinación lineal se pueden calcular, como es habitual, como

$$\phi(x_1, \cdots, x_n) = \langle x_1, \cdots x_n | \varphi \rangle = \langle \Omega | \, \psi(x_1) \cdots \psi(x_n) \, | \varphi \rangle \tag{1.24}$$

1.5. Teoría cuántica de campos

Un campo es el caso límite de una red en la que la distancia ϵ entre los sitios de red tiende a cero ($\epsilon \to 0$) aumentando el número de sitios $N \to \infty$ y manteniendo fija la longitud total $L = N\epsilon$. Al hacer el límite, los operadores de creación y destrucción dan lugar a los operadores de campo:

$$\lim_{\epsilon \to 0} \frac{a_i}{\sqrt{\epsilon}} = \psi_i \tag{1.25}$$

En el caso de que las partículas sean bosones (que es el único caso que consideramos en este trabajo) las relaciones de conmutación entre los operadores de creación y destrucción se traducen en las correspondientes para los campos [9]:

$$[\psi_i, \psi_j^\dagger] = \frac{1}{\epsilon}[a_i, a_j^\dagger] = \frac{\delta_{ij}}{\epsilon} \quad \underset{\epsilon \to 0}{\Longrightarrow} \quad [\psi(x), \psi^\dagger(y)] = \delta(x - y) \tag{1.26}$$

Notar que el factor $\epsilon^{-1/2}$ en la definición (1.25) garantiza que las dimensiones sean correctas, ya que la delta de Kronecker es adimensional mientras que la delta de Dirac tiene dimensiones inversas a las de la variable de integración (longitud en este caso).

Un observable importante es el operador número de partículas, N, definido como

$$N = \int_{-\infty}^{\infty} \mathrm{d}x \psi^\dagger(x) \psi(x) \tag{1.27}$$

Un estado de Fock es autoestado de este operador, y el autovalor correspondiente es la ocupación total del estado.

1.6. El principio variacional

El teorema de Rayleigh-Ritz [10, Complemento E XI] proporciona un método para determinar al estado fundamental de un hamiltoniano H: El valor esperado del hamiltoniano en un estado cualquiera $|\psi\rangle$ es mínimo si ese estado es el fundamental:

$$\langle E \rangle_\psi = \frac{\langle \psi | \, H \, | \psi \rangle}{\langle \psi | \psi \rangle} \geq E_0 \tag{1.28}$$

La estrategia a seguir a la hora de aplicar el principio variacional es elegir una familia de estados que dependan de uno o más parámetros, $|\psi(\alpha)\rangle$ y minimizar el valor esperado del hamiltoniano con respecto a estos parámetros.

Para elegir una clase variacional de funciones, se requiere [11] que sea completa (capaz de reproducir todos los estados del espacio de Hilbert), computacionalmente eficiente (que el número de parámetros no dependa exponencialmente con el tamaño del sistema) y motivada por argumentos físicos. A lo largo de los dos próximos capítulos se mostrará una clase variacional para sistemas unidimensionales que cumple estos requisitos.

2 Estados producto de matrices

En mecánica clásica, el espacio de fases para varias partículas se construye como suma directa de los espacios de fases de cada una de las partículas, y su dimensión crece linealmente con el número de partículas. Por el contrario, en mecánica cuántica los espacios de Hilbert de varias partículas se forman mediante el producto tensorial de los espacios de Hilbert de una partícula. Por lo tanto, su dimensión crece exponencialmente con el número de partículas. Este hecho hace que métodos como el principio variacional, basados en explorar el espacio de estados, están abocados al fracaso debido al gran número de estados posibles incluso en sistemas con unas decenas de partículas.

Sin embargo, la evolución del sistema explora solo una pequeña parte de los estados *a priori* posibles. Así que el problema no es tratar con espacios de dimensión elevada, sino saber seleccionar de ellos los estados relevantes. La ley de área del entrelazamiento permite identificar los estados correspondientes al estado fundamental y primeros excitados, lo que permite acotar enormemente el conjunto de estados en los que buscar.

Hay diversos modelos que explotan las propiedades del entrelazamiento, colectivamente conocidos como redes de tensores. Esto es así porque describen los estados cuánticos como un conjunto de tensores cuyo producto sigue la estructura dictada por el entrelazamiento. El estado producto de matrices es un caso particular, aplicable a redes unidimensionales.

2.1. Tensores

A efectos prácticos, un tensor es una colección multidimensional de números, $A_{\alpha\beta\gamma\ldots}$. El número de índices $\alpha, \beta, \gamma, \cdots$ necesarios para especificar uno de los números es el orden del tensor [12]. Cada índice puede tomar un número de valores discretos posibles, lo que constituye la dimensión de ese índice. La operación básica entre tensores es la contracción de índices (para lo cual los índices contraídos deben tener la misma dimensión D):

$$C_{\beta\gamma\cdots\mu\nu\cdots} = \sum_{\alpha=1}^{D} A_{\alpha\beta\gamma\ldots} B_{\alpha\mu\nu\ldots} \tag{2.1}$$

El caso más elemental es el producto de matrices (una matriz es un tensor de orden 2):

$$C_{\alpha\gamma} = \sum_{\beta=1}^{D} A_{\alpha\beta} B_{\beta\gamma} \tag{2.2}$$

donde la matriz A tiene dimensión d en el índice α y D en el índice β, mientras que B tiene dimensión d en el índice γ, con lo cual C tiene dimensión d en ambos índices: para escribir C hace falta especificar d^2 elementos, y para AB, hay que especificar $2dD$ elementos.

Un tensor se puede representar pictóricamente, mediante la notación de Penrose, como un objeto con un segmento externo por cada índice. De este modo, la contracción de dos tensores se simboliza conectando dos segmentos, como se ilustra en la figura 2.1.

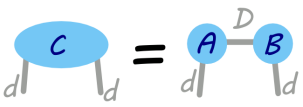

Figura 2.1: Izquierda: Un tensor con dos índices (matriz). Derecha: contracción de dos tensores (producto de matrices).

2.2. Estados cuánticos como tensores

Consideremos una red unidimensional con N sitios. En cada sitio acomodaremos un sistema discreto de d niveles, al que le corresponde un espacio de Hilbert $\mathcal{H}_r, (r = 1, \cdots, N)$ parametrizado por una base de autoestados $\{|i_r\rangle\}, i_r = 1, \cdots, d$. El espacio de Hilbert de la red en su conjunto será

$$\mathcal{H} = \bigotimes_{r=1}^{N} \mathcal{H}_r = \mathcal{H}_1 \otimes \mathcal{H}_2 \otimes \cdots \otimes \mathcal{H}_N \tag{2.3}$$

con base $\{|i_1\rangle \otimes |i_2\rangle \otimes \cdots \otimes |i_N\rangle\}$. En consecuencia, cualquier estado de dicho espacio es de la forma

$$|\psi\rangle = \sum_{i_1, i_2, \cdots, i_N} C_{i_1 i_2 \cdots i_n} |i_1\rangle \otimes |i_2\rangle \otimes \cdots \otimes |i_N\rangle \tag{2.4}$$

Los coeficientes $C_{i_1, i_2, \cdots, i_N}$ necesarios para expresar el estado en términos de los elementos de la base se pueden entender como un tensor de N índices, por lo que contiene d^N elementos. El objetivo es expresar este tensor como la contracción de N tensores de tres índices, de los cuales uno tenga dimensión d y los otros dos dimensión D (dimensión de enlace o *bond dimension*). El estado reescrito de este modo se denomina MPS (*matrix product state*, por sus siglas en inglés).

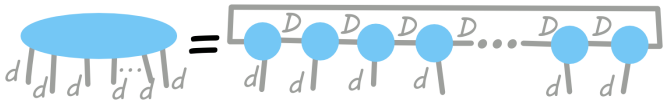

Figura 2.2: Descomposición de un tensor de N índices con dimensión d en la contracción de N tensores de tres índices de dimensiones d y D.

$$|\psi\rangle = \sum_{i_1, \cdots, i_N} \sum_{\alpha, n_1, \cdots, n_{N-1}} A^{(i_1)}_{\alpha n_1} A^{(i_2)}_{n_1 n_2} \cdots A^{(i_N)}_{n_{N-1}\alpha} |i_1\rangle \otimes |i_2\rangle \otimes \cdots \otimes |i_N\rangle \tag{2.5}$$

Construcción mediante descomposición de Schmidt

Se puede demostrar, de forma constructiva, que cualquier estado se puede escribir en la forma MPS. El tensor $C_{i_1 i_2 \cdots i_n}$ se puede considerar como una matriz con solo dos índices: el i_1 y el *índice* resultante de agrupar a todos los demás. Cualquier matriz se puede descomponer mediante la descomposición de valores singulares (SVD)

$$M = U \Sigma V^\dagger \tag{2.6}$$

en el producto de dos matrices unitarias $UU^\dagger = I$ y $VV^\dagger = I$ y una diagonal (no necesariamente cuadrada) Σ con $\Sigma_{ij} = \sigma_i \delta_{ij}$, $\sigma \geq 0$ y $\sum \sigma_i^2 = 1$ por normalización. El número de elementos σ distintos de cero, D_2, es el rango de la matriz. Aplicado al estado cuántico (2.4), resulta en

$$C_{i_1 (i_2 \cdots i_N)} = \sum_{i=1}^{D_2} U_{i_1 k} \sigma_k V^\dagger_{k (i_2 \cdots i_N)} \tag{2.7}$$

Definiendo $|I\rangle = U |i_1\rangle$ y $|J\rangle = V^\dagger |i_2\rangle \otimes \cdots \otimes |i_N\rangle$, se obtiene la **descomposición de Schmidt**:

$$|\psi\rangle = \sum_{i=k}^{D_2} \sigma_k |I_k\rangle \otimes |J_k\rangle \tag{2.8}$$

Iterando el procedimiento de SVD, se llega al estado MPS:

$$C_{i_1 i_2 \cdots i_N} = \sum U_{i_1 k_1} C_{k_1 (i_2 \cdots i_N)} = \sum U_{i_1 k_1} U_{(k_1 i_2) k_2} C_{k_2 (i_3 \cdots i_N)} = \\ = \sum U_{i_1 k_1} U_{(k_1 i_2) k_2} \cdots U_{(k_{N-2} i_{N-1}) k_{N-1}} U_{i_N k_{N-1}} = A^{i_1} A^{i_2} \cdots A^{i_N} \tag{2.9}$$

Para trabajar con condiciones de contorno periódicas (lo cual es recomendable si se va a realizar el límite termodinámico $N \to \infty$), hay que entrelazar el primer sitio de la red con el último contrayendo sus tensores, esto es, haciendo la traza a la expresión anterior:

$$|\psi\rangle = \sum_{i_1, \cdots, i_N} \mathrm{tr}[A^{i_1} A^{i_2} \cdots A^{i_N}] \, |i_1\rangle \otimes |i_2\rangle \otimes \cdots \otimes |i_N\rangle \tag{2.10}$$

Dimensión de enlace y entropía

Hay que notar que la expresión (2.8) no es más que un sistema bipartito, con lo que se puede calcular la entropía de entrelazamiento que existe entre las dos partes a partir de los coeficientes de Schmidt:

$$S = -\sum_{i=k}^{D_2} \sigma_k^2 \log \sigma_k^2 \leq \log D_2 \tag{2.11}$$

Por lo tanto, el rango de la matriz es una medida del entrelazamiento entre los dos subsistemas, y cuanto menor sea el entrelazamiento, menor será la dimensión D_2.

2.3. Utilidad de los estados tensoriales

En conclusión, se ha conseguido reescribir un estado cuántico desde su forma de producto tensorial, para lo que hacen falta d^N elementos, a la forma de estado de producto matricial, en la que hay que especificar NdD^2 elementos. El parámetro clave en esta descripción es D, la dimensión de enlace, que está determinada por el entrelazamiento entre los componentes del sistema. En el caso de que D no crezca exponencialmente con N, el formalismo MPS es una manera eficiente de describir los estados cuánticos.

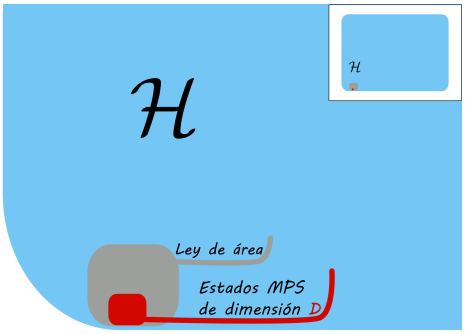

Figura 2.3: El espacio de Hilbert (azul) contiene un número muy elevado de estados, pero de ellos solo unos pocos son físicamente relevantes: los que siguen la ley de área (gris). Eligiendo una dimensión D reducida (rojo), se pueden describir parte de dichos estados.

Los MPS incorporan de manera automática la ley de área. Con un valor de D muy elevado (que dependa exponencialmete con el tamaño del sistema) se puede describir cualquier estado. Al restringirse a valores pequeños, por el contrario, se obtienen solo los estados significativos, y además de forma computacionalmente eficiente.

3 Estados producto de matrices continuos

Los estados de producto de matrices continuos (cMPS por sus siglas en inglés) son la generalización, para campos en una dimensión, de los estados MPS. Se definen por la siguiente expresión [3]:

$$|\chi\rangle = \mathrm{tr}_{aux}\left[\mathcal{P}\mathrm{e}^{\int_0^L \mathrm{d}x[Q(x)\otimes I + R(x)\otimes\psi^\dagger]}\right]|\Omega\rangle \equiv \mathrm{tr}_{aux}U\,|\Omega\rangle \tag{3.1}$$

donde $Q(x)$ y $R(x)$ son matrices de dimensión D que actúan sobre un sistema auxiliar, y que especifican completamente el estado.

3.1. Paso al continuo

Al igual que un campo se puede obtener mediante el paso al continuo de una red atómica, su descripción computacionalmente eficiente, el cMPS (*continuous matrix product state*) se puede obtener mediante el paso al continuo del MPS:

$$|\chi_\epsilon\rangle = \sum_{i_1\cdots i_N} \mathrm{tr}[A^{i_1}A^{i_2}\cdots A^{i_N}](\psi_1^\dagger)^{i_1}\cdots(\psi_N)^{i_N}\,|\Omega\rangle \tag{3.2}$$

Por simplicidad el modelo se puede hacer invariante bajo traslaciones si las matrices A no dependen del sitio de la red, $A^{i_n} = A^n$. La forma de estas matrices se escoje, de acuerdo con [3], para que reproduzcan de forma correcta el límite continuo, empleando las matrices Q y R, ambas de dimensión $D\times D$:

$$A^0 = \mathbb{1} + \epsilon Q \qquad A^n = \frac{\epsilon^n}{n!}R^n \tag{3.3}$$

Comprobemos que realmente se obtiene el resultado esperado. En primer lugar se realiza la discretización del espacio, haciendo $\mathrm{d}x = \epsilon$, $x = n\epsilon$:

$$\begin{aligned}
U &= \mathcal{P}\exp\left[\int_0^L \mathrm{d}x\left(Q\otimes\mathbb{1} + R\otimes\psi^\dagger(n\varepsilon)\right)\right] = \lim_{\varepsilon\to0}\mathcal{P}\exp\left[\sum_{n=0}^{L/\epsilon}\epsilon\left(Q\otimes\mathbb{1} + R\otimes\psi_n^\dagger\right)\right] \\
&= \lim_{\varepsilon\to0}\mathcal{P}\prod_{n=0}^{L/\varepsilon}\exp\left[\epsilon\left(Q\otimes\mathbb{1} + R\otimes\psi_n^\dagger\right)\right] = \lim_{\varepsilon\to0}\mathcal{P}\prod_{n=0}^{L/\varepsilon}\mathrm{e}^{Q\varepsilon}\sum_{m=0}^{\infty}\frac{\varepsilon^m}{m!}R^m\otimes(\psi_n^\dagger)^m \\
&\approx \lim_{\varepsilon\to0}\mathcal{P}\prod_{n=0}^{L/\varepsilon}(1+Q\varepsilon)\left(\mathbb{1} + \sum_{m=1}^{\infty}A^m\otimes(\psi_n^\dagger)^m\right) \approx \lim_{\varepsilon\to0}\mathcal{P}\prod_{n=0}^{L/\varepsilon}\left(A^0 + \sum_{m=1}^{\infty}A^m\otimes(\psi_n^\dagger)^m\right) \\
&= \lim_{\varepsilon\to0}\sum_{i_1,\cdots,i_{L/\varepsilon}} A^{i_1}A^{i_2}\cdots A^{i_{L/\varepsilon}}(\psi_1^\dagger)^{i_1}\cdots(\psi_{L/\varepsilon}^\dagger)^{i_{L/\varepsilon}}
\end{aligned} \tag{3.4}$$

3.2. Expresiones para algunas magnitudes

Los observables del sistema se pueden escribir en función de las matrices Q y R [13]. Los detalles se pueden encontrar en el apéndice A:

$$\langle\chi|\chi\rangle = \mathrm{tr}\left[\mathrm{e}^{TL}\right] \tag{3.5a}$$
$$\langle\psi^\dagger(x)\psi(0)\rangle = \mathrm{tr}\left[\mathrm{e}^{T(L-x)}(R\otimes\mathbb{1})\mathrm{e}^{Tx}(\mathbb{1}\otimes R^\star)\right] \tag{3.5b}$$
$$\langle\psi^\dagger(x)\psi(x)\rangle = \mathrm{tr}\left[\mathrm{e}^{TL}R\otimes R^\star\right] \tag{3.5c}$$
$$\langle\psi^\dagger(0)\psi^\dagger(x)\psi(x)\psi(0)\rangle = \mathrm{tr}\left[\mathrm{e}^{T(L-x)}(R\otimes R^\star)\mathrm{e}^{Tx}(R\otimes R^\star)\right] \tag{3.5d}$$
$$\langle\psi^\dagger(x)\psi^\dagger(x)\psi(x)\psi(x)\rangle = \mathrm{tr}\left[\mathrm{e}^{TL}R^2\otimes(R^\star)^2\right] \tag{3.5e}$$

En todos los casos se ha empleado la matriz de transferencia, definida como

$$T = Q\otimes\mathbb{1} + \mathbb{1}\otimes Q^\star + R\otimes R^\star \tag{3.6}$$

Estas ecuaciones son de vital importancia, ya que permiten relacionar valores esperados para campos (con infinitos grados de libertad), con trazas de matrices de dimensión finita. Por lo tanto, son las herramientas para poder hacer cálculos con un estado cMPS.

3.3. Dinámica disipativa en el sistema auxiliar

Libertad *gauge*

El estado cMPS posee una libertad *gauge* a la hora de elegir las matrices Q y R: queda invariante al hacer la transformación dada por la matriz invertible g:

$$Q \to Q' = gQg^{-1} \qquad R \to R' = gRg^{-1} \tag{3.7}$$

La elección de *gauge* que se suele emplear es la siguiente:

$$Q + Q^\dagger + R^\dagger R = 0 \tag{3.8}$$

que se puede escribir, empleando una matriz hermítica $K = K^\dagger$, como

$$Q = -iK - \frac{1}{2}R^\dagger R \tag{3.9}$$

Con esta elección, el estado (3.1) toma la forma:

$$|\chi\rangle = \mathrm{tr}_{aux}[U(L,0)]\,|\Omega\rangle \qquad U(L,0) = \mathcal{P}\exp\left[-i\int_0^L \mathrm{d}z(K\otimes\mathbb{1} + iR\otimes\psi^\dagger(z) - iR^\dagger\otimes\psi(z))\right] \tag{3.10}$$

Interpretación holográfica

Esta expresión es matemáticamente análoga a la del operador evolución para un campo ψ y un sistema auxiliar (*ancilla*) con D niveles. Esto permitirá reinterpretar el cMPS como un sistema auxiliar con un hamiltoniano K acoplado con un campo que actúa como baño térmico mediante la matriz R, como se ilustra en la figura 3.1. Así pues, la dinámica de un sistema unidimensional queda completamente descrita por la de un sistema puntual, lo que constituye un ejemplo del principio holográfico.

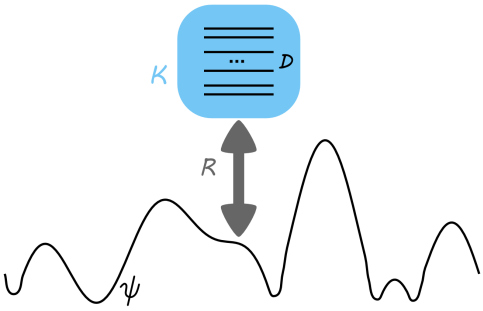

Figura 3.1: El estado cMPS como un sistema de D niveles con hamiltoniano K sometido al baño térmico (campo ψ) mediante la matriz R.

Ecuación de Lindblad

Las expresiones anteriores (3.5) incluyen productos tensoriales en el espacio auxiliar. Para evitarlos, y además hacer patente la dinámica disipativa, se emplea el siguiente isomorfismo:

$$|a\rangle \otimes |b\rangle \Longleftrightarrow |a\rangle \langle b| \tag{3.11}$$

Este isomorfismo se traduce en un isomorfismo entre operadores en $\mathcal{H}\otimes\mathcal{H}$ y superoperadores (es decir, operadores lineales que actúan en el espacio de los operadores de un espacio de Hilbert):

$$(A \otimes B^\star)\, |a\rangle \otimes |b\rangle \Longleftrightarrow A\, |a\rangle \langle b|\, B^\dagger \tag{3.12}$$

Es interesante aplicar el isomorfismo a la matriz de transferencia T (3.6) actuando sobre un vecor $|\rho\rangle$:

$$T\,|\rho\rangle = (Q\otimes \mathbb{1} + \mathbb{1}\otimes Q^\star + R\otimes R^\star)\,|\rho\rangle \Longleftrightarrow \mathsf{T}[\rho] = Q\rho + \rho Q^\dagger + R\rho R^\dagger \tag{3.13}$$

El vector ha pasado a ser una matriz densidad ρ sobre la que opera el superoperador T. Aplicando la expresión (3.9) se obtiene

$$\mathsf{T}[\rho] = -i[K,\rho] + R\rho R^\dagger - \frac{1}{2}\{R^\dagger R, \rho\} \tag{3.14}$$

La expresión (3.14) corresponde a una ecuación de Lindblad [7] que describe la evolución del sistema auxiliar

$$\frac{\partial}{\partial x}\rho(x) = \mathsf{T}[\rho(x)] = -i[K,\rho] + R\rho R^\dagger - \frac{1}{2}\{R^\dagger R, \rho\} \tag{3.15}$$

Si solamente estuviera el primer término, el conmutador, sería una ecuación de Liouville-von Neumann, equivalente a la ecuación de Schrödinger para la matriz densidad de un sistema aislado. Al añadir los términos dependientes de R ya no se está describiendo un sistema aislado sino uno abierto, en contacto con un baño térmico (representado por el campo).

En el límite termodinámico $L\to\infty$, el autovector de T al que corresponda el autovalor de mayor parte real es el dominante en las expresiones (3.5). Para evitar divergencias, esperamos que el mayor autovalor sea cero, lo que equivale a que en (3.15) exista una matriz densidad estacionaria ρ_{ss} tal que $\mathsf{T}[\rho_{ss}] = 0$.

En este límite, las cantidades en (3.5) se pueden escribir en función de la matriz densidad estacionaria:

$$\langle\chi|\chi\rangle = \mathrm{tr}\rho_{ss} = 1 \tag{3.16a}$$

$$\langle\psi^\dagger(x)\psi(0)\rangle = \mathrm{tr}\left[\mathrm{e}^{\mathsf{T}x}(R\rho_{ss})R^\dagger\right] \tag{3.16b}$$

$$\langle\psi^\dagger(x)\psi(x)\rangle = \mathrm{tr}\left[R^\dagger R\rho_{ss}\right] \tag{3.16c}$$

$$\langle\psi^\dagger(0)\psi^\dagger(x)\psi(x)\psi(0)\rangle = \mathrm{tr}\left[R\mathrm{e}^{\mathsf{T}x}(R\rho_{ss}R^\dagger)R^\dagger\right] \tag{3.16d}$$

$$\langle\psi^\dagger(x)\psi^\dagger(x)\psi(x)\psi(x)\rangle = \mathrm{tr}\left[(R^\dagger)^2R^2\rho_{ss}\right] \tag{3.16e}$$

$$\langle\psi^\dagger(x)\left[-\frac{\partial^2}{\partial x^2}\right]\psi(x)\rangle = \mathrm{tr}\left[([Q,R])^\dagger[Q,R]\rho_{ss}\right] \tag{3.16f}$$

Al igual que las relaciones (3.5), estas ecuaciones establecen cómo calcular promedios, y de hecho serán empleadas continuamente en las simulaciones.

3.4. Extensión a campos acoplados

El formalismo de estados de producto de matrices continuos se puede extender para aplicarlo en sistemas de varios campos bosónicos acoplados. Estos campos deberán cumplir las relaciones de conmutación acordes a su estadística:

$$[\psi_\alpha(x),\psi_\beta(x')] = 0 \qquad [\psi_\alpha(x),\psi_\beta^\dagger(x')] = \delta_{\alpha\beta}\delta(x-x') \tag{3.17}$$

La generalización del estado cMPS es inmediata [4]:

$$|\chi\rangle = \mathrm{tr}_{aux}\left[\mathcal{P}\mathrm{e}^{\int_0^L \mathrm{d}x[\tilde{Q}\otimes\mathbb{1}+\sum\tilde{R}_\alpha\otimes\psi_\alpha^\dagger]}\right]|\Omega\rangle \tag{3.18}$$

El sistema está descrito por la matriz $\tilde{Q}$ y las matrices $\tilde{R}_\alpha$: de estas últimas, hay una correspondiente a cada campo ψ_α, y para evitar divergencias, deben heredar las relaciones de conmutación de los campos:

$$[\tilde{R}_\alpha,\tilde{R}_\beta] = 0 \tag{3.19}$$

Al igual que en el caso de un único campo, la evolución temporal se puede describir como una dinámica disipativa mediante una ecuación de Linblad, de modo que las matrices $\tilde{R}_\alpha$ actúan como el acoplamiento del sistema auxiliar con el baño (los campos):

$$\frac{\mathrm{d}}{\mathrm{d}z}\tilde{\rho}(z) = -i[\tilde{K},\tilde{\rho}(z)] + \sum_\alpha \tilde{R}_\alpha\tilde{\rho}\tilde{R}_\alpha^\dagger - \frac{1}{2}\{\tilde{R}_\alpha^\dagger\tilde{R}_\alpha,\tilde{\rho}(z)\} \tag{3.20}$$

El hamiltoniano $\tilde{K}$ se puede relacionar con los parámetros del estado, $\tilde{Q}$ y $\tilde{R}_\alpha$, según la relación:

$$\tilde{Q} = -i\tilde{K} - \frac{1}{2}\sum\tilde{R}_\alpha^\dagger\tilde{R}_\alpha \tag{3.21}$$

Como primer paso vamos a construir las matrices correspondientes a campos no acoplados, a partir de las matrices K y R de un único campo. Para recuperar la dinámica de cada uno de los campos, deberemos exigir que la matriz densidad total sea separable, $\tilde{\rho} = \rho_1\otimes\rho_2$. Cada uno de los campos actúa en su propio espacio con su hamiltoniano K_α y su acoplo con el baño R_α de manera independiente, por lo que

$$\tilde{K} = K_1\otimes\mathbb{1} + \mathbb{1}\otimes K_2 \qquad \tilde{R}_1 = R_1\otimes\mathbb{1} \qquad \tilde{R}_2 = \mathbb{1}\otimes R_2 \tag{3.22}$$

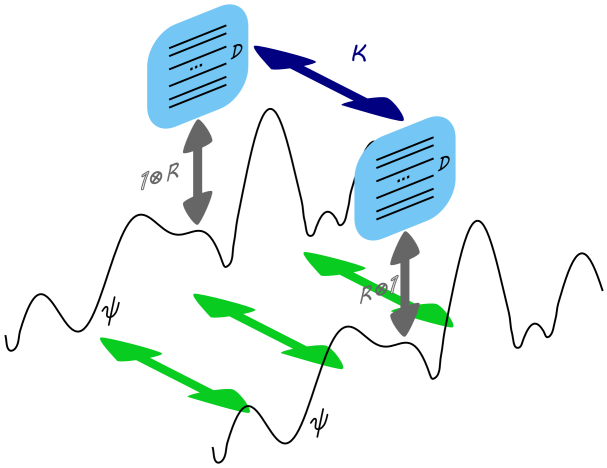

Figura 3.2: En el cMPS para dos campos, el acoplamiento entre ambos se traduce en un acoplamiento entre sus sistemas auxiliares.

Dado que $\tilde{R}_1$ y $\tilde{R}_2$ operan en espacios diferentes, obedecen de manera trivial la relación de conmutación (3.19).

El acoplamiento entre campos se puede introducir adiabáticamente. Al hacerlo, los sistemas auxiliares comienzan a interaccionar, y las soluciones se modifican respecto a las de los campos independientes. En concreto, el estado ya no será separable. Para conseguirlo hay que usar un hamiltoniano $\tilde{K}$ más general:

$$\tilde{K} = K_1 \otimes \mathbb{1} + \mathbb{1} \otimes K_2 + \sum_{p=0}^{P} Z_1^{(p)} \otimes Z_2^{(p)} \tag{3.23}$$

Se han añadido P parejas de matrices hermíticas (para que $\tilde{K}$ también lo sea) y con $Z_1^{(0)} = Z_2^{(0)} = 0$ para reproducir el límite de campos independientes. El número de matrices P es en principio arbitrario, pero el número óptimo aumentará al incrementar el acoplamiento. Las matrices $\tilde{R}_1$ y $\tilde{R}_2$ no se modifican respecto del caso sin perturbar.

Empleando una dimensión de enlace D, el número de parámetros variacionales para describir los dos campos asciende a $(4 + 2P)D^2$.

4 Metodología

El presente trabajo está dedicado a la simulación computacional de campos cuánticos. En consecuencia, la labor de investigación correspondiente se ha realizado mediante la elaboración y ejecución de los códigos informáticos necesarios. La elaboración del código se estructuró como sigue:

- En primer lugar se estudió un único campo. Para ello se empleó un código diseñado por el alumno (Apéndice B) en el lenguaje de programación `C`. La elección del lenguaje se debe a su rapidez y flexibilidad, y también porque el alumno está familiarizado por su utilización en otras asignaturas del grado. El código se basa en el que presenta [11] (en el lenguaje `MATLAB`), incluyendo mejoras para incrementar su eficiencia computacional. La ejecución del programa se realizó en el ordenador personal del alumno, con el sistema operativo *Ubuntu* (Linux) y el compilador `gcc`.

- Después se realizó la simulación de dos campos acoplados. En esta etapa, la mayor exigencia de potencia de cálculo marcó la metodología necesaria. Las rutinas de optimización en `C` se mostraron insuficientes al incrementar el número de parámetros variacionales, con lo que en su lugar se empleó `MATLAB` debido a su potente rutina `fmincon`. Para hacer frente al elevado tiempo de cálculo, la ejecución se realizó en el *cluster* del Departamento de Física de la Materia Condensada de la Universidad de Zaragoza (con sistema operativo Linux). El código fue elaborado para su investigación por uno de los directores del trabajo, Fernando Quijandría, y modificado para adaptarse al problema específico por el alumno.

4.1. Algoritmo para un solo campo

La idea del algoritmo es tomar como parámetros variacionales las matrices K y R, ambas de dimensión M, y con ellas calcular la matriz densidad ρ_{ss} según (3.14). Esta matriz se emplea para calcular la energía del modelo de Lieb-Liniger (5.1) aplicando los resultados (3.16f) y (3.16e):

$$\langle E\rangle = \langle\psi^\dagger(x)\left[-\frac{\partial^2}{\partial x^2}\right]\psi(x)\rangle + c\langle\psi^\dagger(x)\psi^\dagger(x)\psi(x)\psi(x)\rangle = \mathrm{tr}\left[([Q,R])^\dagger[Q,R]\rho_{ss}\right] + \mathrm{tr}\left[c(R^\dagger)^2R^2\rho_{ss}\right] \tag{4.1}$$

Los valores de energía así calculados se emplean para su optimización numérica según (1.28), tomando como condición de minimización el valor de N (recordemos que para el modelo de Lieb-Liniger es una constante del movimiento), que se calcula de acuerdo con su definición (1.27) y el resultado (3.16c):

$$\langle N\rangle = \langle\psi^\dagger(x)\psi(x)\rangle = \mathrm{tr}\left[R^\dagger R\rho_{ss}\right] \tag{4.2}$$

Así pues, al fijar la condición $\langle N\rangle = \rho$ se está restringiendo la optimización al subespacio de estados con densidad de partículas ρ. El proceso de minimización se inicia con unas matrices K y R aleatorias y se continúa hasta que el valor de la energía converja a un mínimo.

Una vez calculadas las matrices K y R, y con ellas la matriz densidad ρ_{ss}, es posible calcular funciones de correlación empleando (3.16b) y (3.16d).

Modificaciones del código

Respecto del código original, se han realizado algunas modificaciones:

- El programa, originariamente en `MATLAB`, se ha reescrito en `C`. Para ello ha sido necesaria la utilización de librerias: para las operaciones matriciales la *GNU Scientific Library* [14] y para las rutinas de optimización la librería *NLOpt* [15].

- Se ha modificado el método usado para calcular el estado estacionario del *Liouvilliano*: en el original se usa el *inverse power method*, pero dado que las matrices empleadas son singulares, esto conlleva problemas de estabilidad numérica. En su lugar, se ha empleado la descomposición QR: El *Liouvilliano* se expresa como $\mathcal{L} = A \cdot B$, siendo A unitaria (y por tanto regular) y B rectangular superior. El estado estacionario ρ se encuentra resolviendo por *backsubstitution* el sistema $B\rho = 0$. Este cambio ofrece una ventaja adicional, ya que requiere un menor número de operaciones: Al tratar con *superoperadores* (matrices de dimensión $D^2 \times D^2$), las multiplicaciones necesarias para calcular el *Liouvilliano* emplean $\mathcal{O}(D^6)$ operaciones. El *inverse power method* requiere efectuar una inversión ($\mathcal{O}(D^6)$) y un número, a priori desconocido, de multiplicaciones ($\mathcal{O}(D^6)$); mientras que el método alternativo ejecuta una descomposición QR ($\mathcal{O}(D^6)$) y una *backsubstitution* ($\mathcal{O}(D^4)$).

- El código original emplea para la optimización sujeta a restricciones mediante la función `fmincon` de `MATLAB`, que opera a modo de caja negra. Este enfoque, aunque cómodo, no está exento de riesgos, ya que un conocimiento más detallado del funcionamiento interno puede usarse para mejorar el rendimiento: en este caso, se puede evitar calcular varias veces la matriz ρ_{ss} a partir de los mismos $\{K, R\}$. Esto es una gran ventaja, ya que éste es precisamente la etapa más exigente computacionalmente.

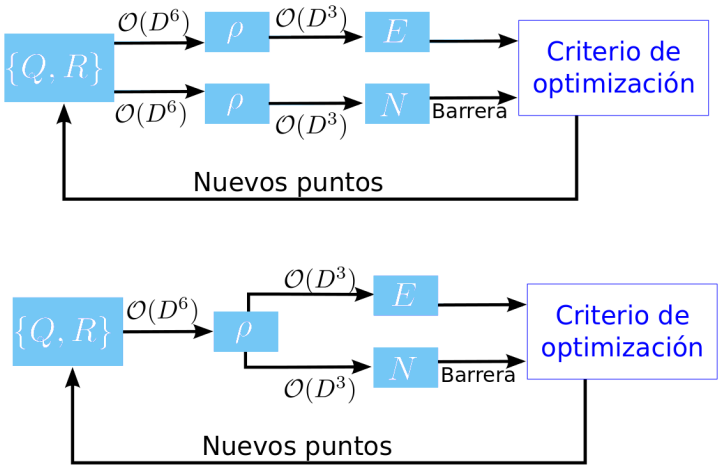

Figura 4.1: Esquema de operación de la fase de optimización en el código original (arriba) y en el actual (abajo).

El proceso de optimización

La optimización sujeta a restricciones se realiza mediante el llamado *método de la barrera* (`fmincon` lo utiliza, aunque los detalles de su implementación permanezcan obscuros y ultra-secretos). Para minimizar la función $f(x)$ obedeciendo la condición $g(x) = 0$, el método de la barrera minimiza la función

$$F(x) = f(x) + I(g(x)) \tag{4.3}$$

La función $I(x)$ es la barrera, tal que $I(0) \ll I(x),\ |x| > 0$. Físicamente esto se puede interpretar como añadir a la energía un potencial que atrae al sistema hacia las regiones del espacio de fases que verifican $g(x) = 0$. Una primera idea para elegir la barrera es $I(x) = -\delta(x)$. Sin embargo, esto no es conveniente, ya que partiendo de condiciones iniciales aleatorias es prácticamente imposible llegar a la región que verifica las condiciones, mientras que partiendo desde un punto válido, el movimiento por el espacio de fases está muy limitado impidiendo llegar a otros mínimos locales.

Una solución es emplear como barrera una función con forma de *embudo*: partiendo de cualquier punto arbitrario se acaba rápidamente en la región de interés, permitiendo una exploración de una región mayor del espacio de fases. Algunas de las funciones que se han probado son $I(x) = |x|$, $I(x) = \tan^{-1}(x)$ y $I(x) = \tanh^{-1}(x)$.

En la literatura matemática, los problemas de optimización sujeto a restricciones de igualdad atraen mucha menos atención que las restricciones de desigualdad, ya que estos últimos son más fáciles de tratar. Usando este enfoque, se puede transformar el problema actual en uno más sencillo computacionalmente teniendo en cuenta que la energía es una función creciente con el número de partículas: así, fijando el número mínimo de partículas mediante la barrera, la propia función a minimizar se encargará de acotarlo superiormente. La opción favorita es la barrera logarítmica, $I(x) = -t^{-1}\log(x),\ t \to \infty$.

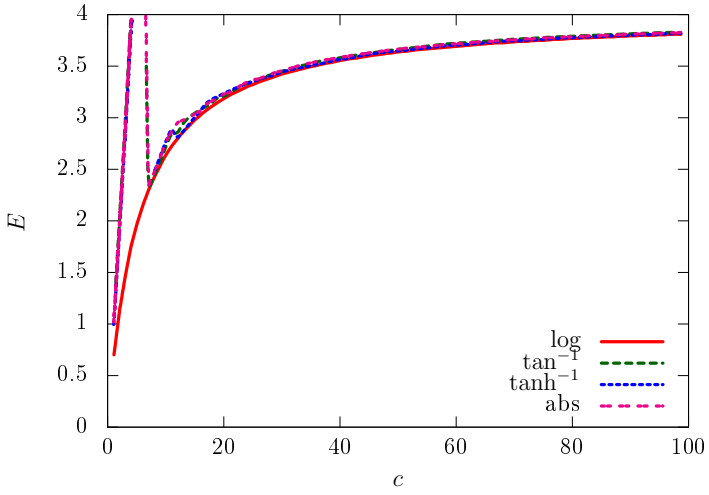

Figura 4.2: Resultados para $D = 2$ obtenidos con diferentes barreras en la optimización.

De entre todos los métodos expuestos anteriormente, se encuentra que es preferible el de la barrera logarítmica, tanto por ofrecer resultados algo mejores como por requerir de menos tiempo de ejecución.

El proceso de optimización exhibe una marcada dependencia a los puntos iniciales suministrados. Para conseguir los mejores resultados, al calcular la energía correspondiente a un valor de c se comienza con el estado óptimo calculado para otro valor de c próximo. Aun así, es necesario hacer varias rondas aumentando y disminuyendo sucesivamente el valor de c para que el resultado converja.

Además, al aumentar la dimensión de enlace D se está incrementando el número de parámetros variacionales, por lo que obtener una buena convergencia en la optimización se hace más complicado. Por ello es conveniente poder construir una solución en dimensión D partiendo de la solución en dimensión $D-1$. Esto se consigue empleando las matrices $K^{(D)}$, $R^{(D)}$ obtenidas al añadir una fila y columna de ceros a las matrices $K^{(D-1)}$ y $R^{(D-1)}$. Es inmediato comprobar que la matriz densidad estacionaria que se obtiene $\rho_{ss}^{(D)}$ es igual a $\rho_{SS}^{(D-1)}$ con la fila y columna de ceros añadidas,, y que la densidad de partículas y la energía que se calculan con ambas también coinciden. Por lo tanto, al ir subiendo en dimensión es conveniente emplear las matrices calculadas en las dimensiones anteriores como valor inicial para la minimización.

4.2. Algoritmo para dos campos acoplados

La idea tras el algoritmo es similar a la que se emplea para un único campo, aunque presenta algunas peculiaridades.

El acoplamiento entre los campos, a nivel de los operadores en el sistema auxiliar, se realizó de forma perturbativa: la matriz K se construye a partir de hamiltonianos para campos independientes, añadiéndoles P parejas de matrices Z (3.23). Esta misma lógica se emplea en la simulación: en primer lugar se simula un único campo con lo que se obtienen sus matrices K y R. A continuación se crean las matrices $\tilde{K} = K \otimes \mathbb{1} + \mathbb{1} \otimes K$, $\tilde{R}_1 = R \otimes \mathbb{1}$ y $\tilde{R}_2 = \mathbb{1} \otimes R$, que se emplean para simular los dos campos sin introducir aún el acoplamiento entre ellos. Finalmente se añaden al hamiltoniano $\tilde{K}$ tantos pares de matrices Z como sea necesario y se introduce el acoplamiento, de forma adiabática.

El objetivo es poder reproducir una transición de fase. De ser así, las magnitudes deberán cambiar bruscamente. Como es complicado reproducir este cambio en un proceso de optimización, en la zona próxima a la transición trataremos de *ayudar* al programa a que haga variaciones más grandes: antes de cada paso de minimización, a cada elemento de las matrices K, R y Z le sumaremos o restaremos un número aleatorio, cuyo valor es como máximo el 10 % del elemento de matriz correspondiente.

5 Modelos y resultados

5.1. Modelo de Lieb-Liniger

Modelo teórico

El modelo de Lieb-Liniger describe un sistema de partículas bosónicas no relativistas en una dimensión que exhiben interacción de contacto (potencial delta de Dirac). Es por tanto el modelo con interacción más sencillo posible. El hamiltoniano correspondiente (con unidades en las que $\hbar = 2m = 1$) es

$$H = \int_{-\infty}^{\infty} \mathrm{d}x \left(\frac{\mathrm{d}\psi^\dagger(x)}{\mathrm{d}x} \frac{\mathrm{d}\psi(x)}{\mathrm{d}x} + c\psi^\dagger(x)\psi^\dagger(x)\psi(x)\psi(x) \right) \tag{5.1}$$

La interacción es repulsiva haciendo $c > 0$. El número de partículas es una cantidad conservada para este hamiltoniano, $[H, N] = 0$. Por lo tanto, partiendo de un estado en el subespacio de Fock con N_0 partículas, la evolución dada por el hamiltoniano no saca al sistema de ese subespacio.

El modelo de Lieb-Liniger se puede resolver exactamente mediante el *ansatz* de Bethe [16][17]. Si la densidad de partículas es ρ, se puede definir un acoplamiento adimensional

$$\gamma = \frac{c}{\rho} \tag{5.2}$$

La energía del estado fundamental se puede escribir en términos de la densidad y de la densidad de energía e, que depende solo del acoplamiento adimensional:

$$E_{LL}(c, \rho) = \rho^3 e(\gamma) \tag{5.3}$$

La función $e(\gamma)$ no tiene una expresión analítica, aunque se pueden emplear las fórmulas asintóticas:

$$\lim_{\gamma \to 0} e(\gamma) = \gamma \left(1 - \frac{4}{3\pi}\sqrt{\gamma} \right) \to \gamma \tag{5.4a}$$

$$\lim_{\gamma \to \infty} e(\gamma) = \frac{1}{3}\pi^2 \left(\frac{\gamma}{\gamma + 2} \right)^2 \to \frac{1}{3}\pi^2 \tag{5.4b}$$

El primer límite es la aproximación de campo medio. El límite de acoplamiento infinito se conoce como aproximación de Tonks-Girardeau, y corresponde a tener una única partícula por sitio.

En la actualidad se pueden hacer sistemas físicos que reproduzcan el modelo de Lieb-Liniger, por ejemplo en condensados atrapados en trampas ópticas cuasiunidimensionales [18].

Energía

El primer objetivo de las simulaciones es comprobar si los resultados que producen son correctos. Para ello, se utiliza un modelo sencillo, el modelo de Lieb-Liniger, y se comparan los resultados con los publicados previamente, tanto con el mismo método [3] [11], como con los resultados analíticos [16] [17].

En esta simulación se calcula la energía del nivel fundamental del modelo de Lieb-Liniger (5.1), en función del parámetro de la interacción c. Para ello se trabaja con la densidad de partículas fija en $\langle N \rangle = \rho = 1$, y se emplean distintos valores de la dimensión de enlace D. El resultado se muestra en la figura 5.1a

Para poder cuantificar la calidad de los datos, se realiza el ajuste de los resultados para $c > 50$ a una dependencia del tipo $E(c) = A\left(\frac{c}{c+B}\right)^2$, obteniendo los parámetros de ajuste recogidos en

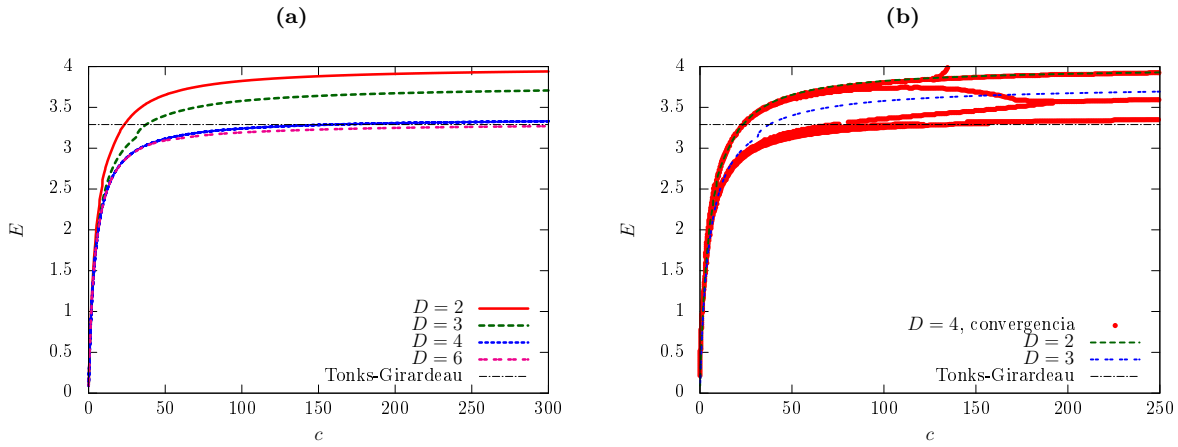

Figura 5.1: (a) Densidad de energía E en el modelo Lieb-Liniger en función de la intensidad c. (b) Proceso de convergencia hasta el resultado óptimo con $D = 4$. Se observa que el sistema permanece bastante tiempo en las curvas que se obtenían con $D = 2$ y $D = 3$.

D	A	B
2	3.97921	2.18811
3	3.65069	2.23825
4	3.3646	1.87705
6	3.31124	1.96327

Tabla 5.1: Ajustes de los datos a la forma funcional asintótica.

la tabla 5.1. Se observa que con $D = 2$ y $D = 3$ no es posible recrear los resultados analíticos, pero con muy poco esfuerzo computacional adicional, a partir de $D = 4$ los resultados obtenidos difieren en un 2 % de los que cabría esperar y convergen rápidamente hacia la solución exacta. Este es el comportamiento esperado, ya que incluso con dimensiones de enlace muy bajas se puede reproducir fielmente los estados relevantes.

Es interesante ver lo que ocurre al simular un campo en una dimensión relativamente grande ($D \geq 4$) partiendo de valores iniciales aleatorios, como se ilustra en la figura 5.1b. En un principio no se obtienen las curvas de la figura 5.1a, sino que hacen falta varios *barridos* aumentando y disminuyendo secuencialmente c para que la solución converja. Mientras tanto, los puntos que se obtienen no aparecen en cualquier sitio: solo están sobre las curvas obtenidas para $D = 2$ y $D = 3$. Esto puede indicar que se están observando distintos niveles del espectro discreto de energías.

Introducción de potencial químico

Efectuando la transformada de Legendre respecto del número de partículas se obtiene el hamiltoniano gran canónico

$$H = \int_{-\infty}^{\infty} \mathrm{d}x \left(\frac{\mathrm{d}\Psi^\dagger(x)}{\mathrm{d}x} \frac{\mathrm{d}\Psi(x)}{\mathrm{d}x} + c\Psi^\dagger(x)\Psi^\dagger(x)\Psi(x)\Psi(x) - \mu\Psi^\dagger(x)\Psi(x) \right) \tag{5.5}$$

Las propiedades termodinámicas [17] en el límite de Tonks-Girardeau están dadas por las expresión:

$$\mu = (\pi\rho)^2 \tag{5.6}$$

En la gráfica 5.2a se observa cómo el modelo es capaz de reproducir esta dependencia, incluso con valores bajos de D.

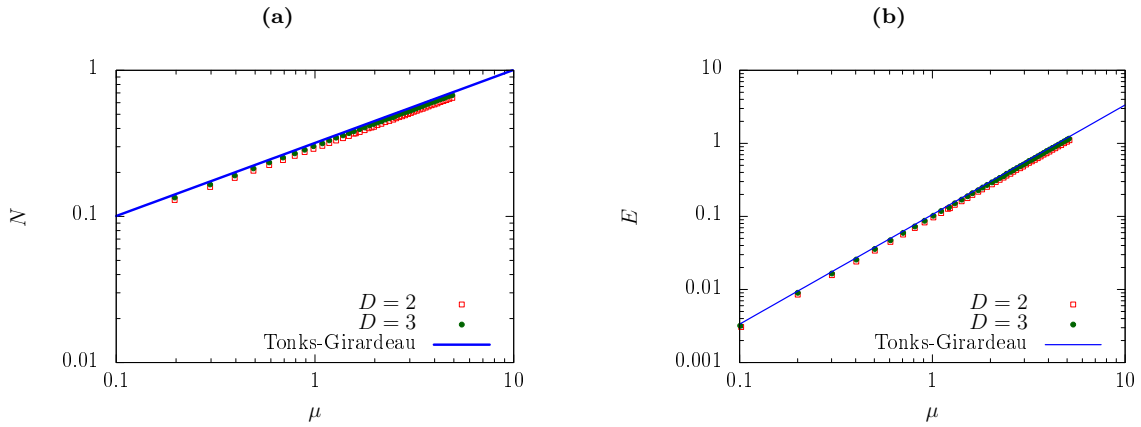

Figura 5.2: Densidad de número de partículas (a) y energía (b) en el modelo de Lieb-Liniger con potencial químico μ en el límite de Tonks-Girardeau.

Sustituyendo en la expresión para la energía en el límite de Tonks-Girardeau (5.4b), se obtiene

$$E = \frac{\pi^2}{3}\rho^3 = \frac{\pi^2}{3}\frac{\mu^{3/2}}{\pi^3} = \frac{1}{3\pi}\mu^{3/2} \tag{5.7}$$

En la figura 5.2b se aprecia cómo la energía como el número de partículas ajusta bien a la previsión teórica, incluso con valores bajos de D.

Funciones de correlación

Las últimas magnitudes que calcularemos con el modelo con un único campo son las funciones de correlación densidad-densidad.

Las funciones de correlación densidad-densidad, como la gráfica 5.3, se ajustan satisfactoriamente a los resultados teóricos. En concreto, para valores elevados de x las densidades de los dos puntos son independientes y

$$\langle n(x)n(0)\rangle = \langle n(x)\rangle\langle n(0)\rangle = \rho^2 \tag{5.8}$$

Para distancias cortas, al aproximarse al régimen de Tonks-Girardeau los bosones son prácticamente impenetrables, por lo que la función de correlación tiende a cero. Si por el contrario la constante c es pequeña, las partículas apenas interaccionan y su comportamiento se asemeja al de un gas ideal de bosones: las densidades están más próximas a ser independientes. Los resultados obtenidos reproducen correctamente las funciones de correlación teóricas [19].

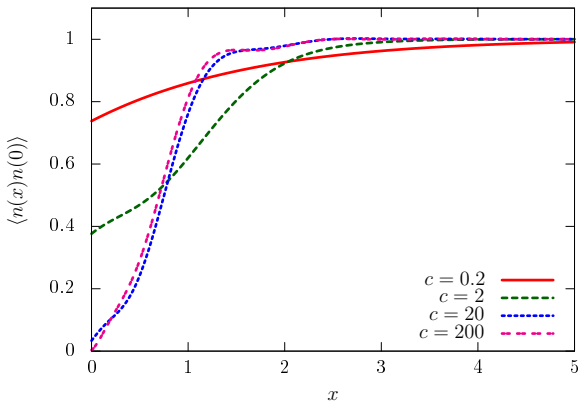

Figura 5.3: Funciones de correlación densidad-densidad para varios valores de c en dimensión $D = 4$.

5.2. Dos campos acoplados

Modelo teórico y transición de fase

El modelo que emplearemos son dos campos de Lieb-Liniger con un acoplamiento dependiente de la densidad. La intensidad del acoplamiento es g.

$$H = \int_0^L \mathrm{d}x \sum_{\alpha=1,2} \partial_x \psi_\alpha^\dagger \partial_x \psi_\alpha + c\psi_\alpha^\dagger \psi_\alpha^\dagger \psi_\alpha \psi_\alpha + g\psi_1^\dagger \psi_2^\dagger \psi_1 \psi_2 = H_{LL} \otimes \mathbb{1} + \mathbb{1} \otimes H_{LL} + g n_1 n_2 \quad (5.9)$$

El hamiltoniano conmuta con la densidad de partículas de cada especie por separado, $[H, \rho_1] = [H, \rho_2] = 0$. Esto nos permite mantener las densidades fijas y emplearlas como restricciones para la minimización. Mantendremos la densidad total $\rho = \rho_1 + \rho_2$ fija.

En el caso de dos campos independientes, $g = 0$, el término dominante en el hamiltoniano corresponde a la repulsión entre partículas de la misma especie. Como consecuencia, la situación de mínima energía se produce cuando hay la menor densidad de partículas posible de cada especie, es decir, $\rho_1 = \rho_2 = \frac{1}{2}\rho$. En el caso de interacción elevada, $c \to \infty$:

$$E = \frac{\pi^2}{3}({\rho_1}^3 + {\rho_2}^3) \quad (5.10)$$

$$\mathrm{d}E = \pi^3({\rho_1}^2 \mathrm{d}\rho_1 + {\rho_2}^3 \mathrm{d}\rho_2) = \pi^3({\rho_1}^2 - {\rho_2}^2)\mathrm{d}\rho_1 = 0 \qquad \rho_1 = \rho_2 = \frac{1}{2}\rho \quad (5.11)$$

y en el caso de $c \to 0$:

$$E = {\rho_1}^3 \gamma_1 + {\rho_2}^3 \gamma_2 = c({\rho_1}^2 + {\rho_2}^2) \quad (5.12)$$

$$\mathrm{d}E = 2c(\rho_1 \mathrm{d}\rho_1 + \rho_2 \mathrm{d}\rho_2) = 2c(\rho_1 - \rho_2)\mathrm{d}\rho_1 = 0 \qquad \rho_1 = \rho_2 = \frac{1}{2} \quad (5.13)$$

Por el contrario, en el límite de acoplo elevado entre los dos campos elevados, $g \to \infty$, el hamiltoniano se puede aproximar como

$$H \approx g n_1 n_2 \quad (5.14)$$

En estas condiciones, domina la repulsión entre las dos especies. La mínima energía corresponderá, por lo tanto, a que solo esté presente una de las dos especies: $\rho_1 = \rho, \rho_2 = 0$ o $\rho_1 = 0, \rho_2 = \rho$.

En el rango intermedio de acoplo, se producirá la transición de fase entre ambos comportamientos. Para estimar el punto en el que tendrá lugar, aplicamos la aproximación de Thomas-Fermi, consistente en despreciar el término cinético del hamiltoniano, con lo que el sistema es resoluble

$$H = c({\rho_1}^2 + {\rho_2}^2) + g\rho_1\rho_2 = c{\rho_1}^2 + c(\rho - \rho_1)^2 + g\rho_1(\rho - \rho_1) \quad (5.15)$$

$$\frac{\mathrm{d}H}{\mathrm{d}\rho_1} = 2c\rho_1 + 2c(\rho_1 - \rho) + g\rho - 2g\rho_1 = (2c - g)(2\rho_1 - \rho) = 0 \quad (5.16)$$

Se observa que $\rho_1 = \frac{1}{2}\rho$ siempre hace la energía extremal. Para comprobar si es un máximo o un mínimo, recurrimos a la segunda derivada

$$\frac{\mathrm{d}^2 E}{\mathrm{d}{\rho_1}^2} = 2(2c - g) \quad (5.17)$$

En consecuencia, la transición de fase se produce en $g^* = 2c$. Para $g < g^*$, el estado de mínima energía es que los dos campos tengan la misma densidad, mientras que en $g > g^*$, se produce la separación de las dos especies [20].

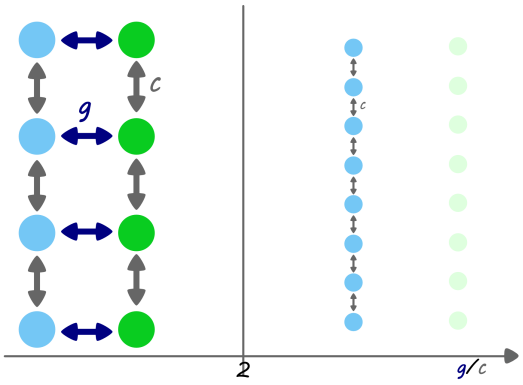

Figura 5.4: Transición de fase entre dos campos con igual población (acoplamiento g bajo, izquierda) y poblamiento de un único campo (g elevado, derecha).

Cabe preguntarse cuál será el estado cuántico del sistema $|\varphi(\rho, g)\rangle$ en las dos fases. En primer lugar, con $g = 0$, simplemente se tienen dos campos de Lieb-Liniger independientes con densidad $\rho/2$:

$$|\varphi(\rho, g = 0)\rangle = |\chi(\rho/2)\rangle \otimes |\chi(\rho/2)\rangle \tag{5.18}$$

La energía correspondiente a este estado será

$$E(\rho, g = 0) = \langle\varphi(\rho, g = 0)| H |\varphi(\rho, g = 0)\rangle = 2E_{LL}(\rho/2) \tag{5.19}$$

Para valores de g pequeños, el término de acoplamiento se puede considerar como una perturbación respecto al hamiltoniano de los dos campos independientes, por lo que la corrección a la energía se puede calcular por teoría de perturbaciones:

$$E(\rho, g) \approx 2E_{LL}(\rho/2) + \langle\varphi(\rho, g = 0)| gn_1n_2 |\varphi(\rho, g = 0)\rangle = 2E_{LL}(\rho/2) + g\frac{\rho^2}{4} \tag{5.20}$$

Una vez superada la transición, uno de los campos está en el estado del vacío, y el otro en el estado de un Lieb-Liniger sencillo de densidad ρ. El estado más general para describir el sistema es una superposición de las dos posibles elecciones:

$$|\varphi(\rho, g)\rangle = \alpha |\chi(\rho)\rangle \otimes |\Omega\rangle + \beta |\Omega\rangle \otimes |\chi(\rho)\rangle \qquad |\alpha|^2 + |\beta|^2 = 1 \tag{5.21}$$

Todos estos estados están degenerados en energía:

$$n_1n_2 |\chi(\rho)\rangle \otimes |\Omega\rangle = n_1n_2 |\Omega\rangle \otimes |\chi(\rho)\rangle = 0 \qquad E(\rho, g) = \langle\varphi(\rho, g)| H |\varphi(\rho, g)\rangle = E_{LL}(\rho) \tag{5.22}$$

Sin embargo, difieren en el valor esperado de la densidad de cada uno de los campos:

$$\rho_1 = \langle\varphi(\rho, g)| n_1 |\varphi(\rho, g)\rangle = |\alpha|^2\rho \qquad \rho_2 = \langle\varphi(\rho, g)| n_2 |\varphi(\rho, g)\rangle = |\beta|^2\rho \tag{5.23}$$

En las simulaciones de dos campos de Lieb-Liniger acoplados, nos centraremos en intentar localizar y caracterizar la transición de fase que se produce al variar el parámetro de acoplo entre campos g. Para ello mantendremos la densidad total del sistema fija, y emplearemos $P = 4$ pares de matrices Z para representar el acoplamiento a nivel del sistema auxiliar.

Energía

Un indicador importante de la transición es el comportamiento de la energía en función del acoplo g, representado en la figura 5.5. Con acoplos elevados, solo existe población en uno de los campos, por lo que la energía no puede depender de la intensidad de esta interacción: su valor

debería ser el de la energía de un modelo de Lieb-Liniger con densidad ρ. Por el contrario, con acoplos bajos sí que existen interacciones interespecie, y la energía mostrará una dependencia con g. Aunque en principio no conocemos cómo es esa dependencia (además de monótonamente creciente), sí que podemos saber que, con $g = 0$, la energía de cada uno de los campos es la de un modelo de Lieb-Liniger con densidad $\rho/2$.

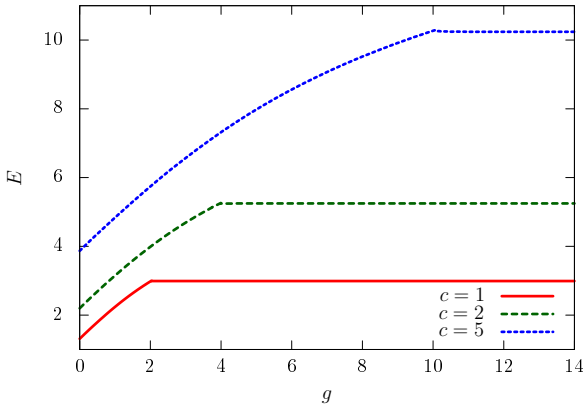

Figura 5.5: Energía total. Para la simulación se ha usado $D = 4$ y $\rho = 2$, y varios parámetros de la interacción de Lieb-Liniger c.

Tal y como se esperaba, la energía va aumentando al incrementar g hasta que se produce la transición en $g = 2c$. A partir de ese punto, la energía se mantiene siempre constante.

Para comprobar si, además de cualitativamente, la simulación también se comporta de forma adecuada cuantitativamente, podemos comparar los valores numéricos de la energía en los casos extremos con los que se obtendrían con el modelo de Lieb-Liniger sin acoplar en la simulación previa. Los resultados se recogen en la tabla 5.2

c	Simulación $g = 0$	Lieb-Liniger $2E(\rho/2)$	Simulación $g > 2c$	Lieb-Liniger $E(\rho)$
1	1.31	1.29	2.99	2.98
2	2.20	2.13	5.25	5.17
5	3.87	3.59	10.24	9.81

Tabla 5.2: Comparación entre la energía para los dos campos acoplados (*simulación*) y la situación correspondiente con un solo campo (*Lieb-Liniger*).

El grado de acuerdo entre las simulaciones dos campos y el resultado previsto con las simulaciones de un único campo es bastante bueno, en especial cuando c es bajo.

Densidad

Otro de los aspectos que permiten distinguir la transición de fases es la densidad que presentan ambos campos, representada en 5.6a

En todas las simulaciones realizadas se encuentra que, en el régimen de $g > 2c$, se cumple que $|\Delta\rho| = |\rho_1 - \rho_2| = \rho$. Así pues, aunque el sistema en principio pueda encontrarse en cualquier superposición de los estados con uno de los campos despoblados (5.21), solo se observan los casos de $\alpha = 1$ y $\alpha = 0$. Esto se puede justificar porque son los estados con menor entrelazamiento, y por tanto los que se describen con mayor facilidad por el cMPS. Añadiendo más restricciones a la relación que deben guardar las densidades de los dos campos se podría forzar a otros valores de $|\Delta\rho|$ con la misma energía.

En la gráfica 5.6b se aprecia que, para distintos valores de c, en el valor $g = 2c$ se produce una variación muy brusca entre las dos fases, lo cual es una señal inequívoca de una transición

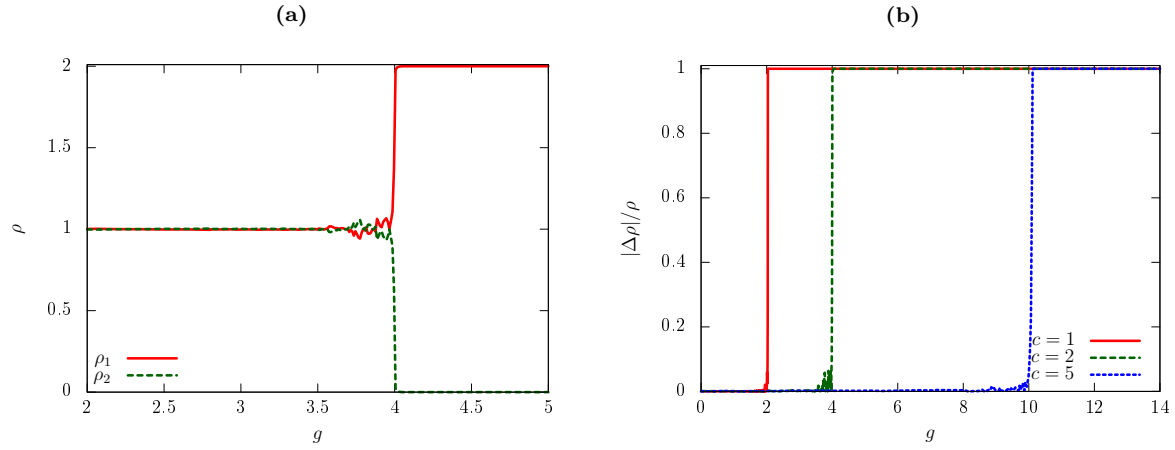

Figura 5.6: (a) Población en los dos campos. Los parámetros utilizados son $D = 4$, $c = 2$, $\rho = 2$ y $P = 3$. (b) Diferencia de población entre los campos. Se ha usado $D = 4$ y $\rho = 2$, y varios valores de c.

de fase. Para valores de g algo menores se observa una cierta fluctuación respecto de $\Delta\rho = 0$. Esto es la consecuencia del método empleado para la simulación, en el que a las matrices que describen el estado se les ha añadido una componente aleatoria.

Funciones de correlación

La diferencia cualitativa de comportamiento también se manifiesta en las funciones de correlación densidad-densidad. Para $g = 0$ (figura 5.7a), los dos campos son independientes, lo que se ve claramente porque $\langle n_1(x)n_2(0)\rangle = \langle n_1(x)\rangle\langle n_2(0)\rangle = (\rho/2)^2$. Además, las funciones de correlación para ambos campos son idénticas entre sí y a la figura 5.3.

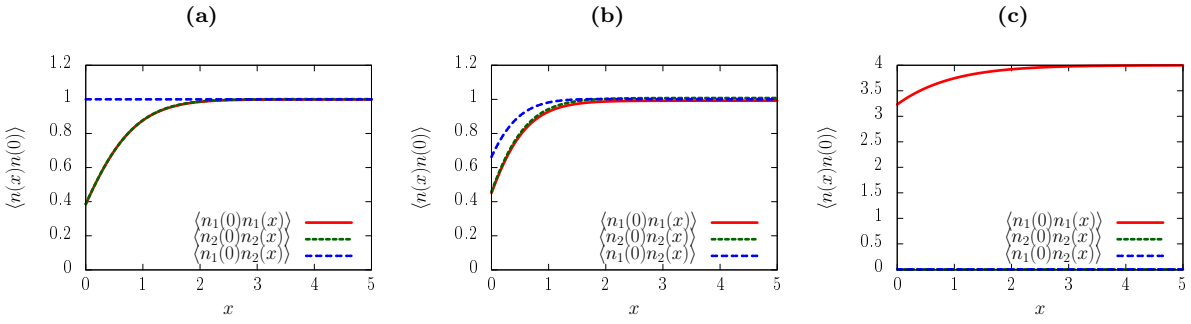

Figura 5.7: Funciones de correlación con $g = 0$ (a), g =2.8 (b) y g =4.4 (c). Cálculos realizados con $D = 4$, $P = 3$, $c = 2$ $\rho = 2$.

Para $g > 2c$, como se observa en la figura 5.7c, uno de los dos campos tiene densidad nula, como demuestra el hecho de que su función de correlación sea nula a todas las distancias. Al otro campo le corresponde la función de correlación de un campo de Lieb-Liniger simple con densidad doble.

Por último, también se han calculado las funciones de correlación en el caso $g < 2c$, representadas en la figura 5.7b. Es interesante notar que en estas condiciones las densidades de los dos campos no son independientes entre sí, $\langle n_1(x)n_2(0)\rangle \neq \langle n_1(x)\rangle\langle n_2(0)\rangle$. Además, las correlaciones de ambos campos son prácticamente iguales, lo que indica que el estado es simétrico bajo la inversión de los dos campos.

Conclusiones

En la primera parte de la memoria del trabajo hemos seguido los pasos que han llevado a la elaboración del modelo teórico que lo sustenta. Se ha argumentado que en los sistemas físicos de la materia condensada la simulación por *fuerza bruta* es una aproximación ineficiente e inútil, y que una buena solución es la selección de los estados físicamente relevantes. Para ello se ha visto cómo emplear los estados producto de matrices para sistemas unidimensionales discretos, y los estado producto de matrices continuos para campos unidimensionales. Se ha comprobado también que es posible describir de este modo varios campos que interaccionan entre sí, y cómo esta interacción puede dar lugar a una transición de fase.

En la segunda parte de la memoria se ha dado el salto del papel al ordenador, y se ha expuesto el proceso de simulación numérica de campos cuánticos guiada por los métodos estudiados. Se han realizado simulaciones con un único campo cuántico, comprobando que se obtienen los resultados correctos al calcular en el modelo de Lieb-Liniger la energía, las funciones de correlación, y las propiedades termodinámicas en el conjunto gran canónico. Una vez verificada la solvencia y eficacia computacional del método, se ha procedido a la obtención de resultados novedosos en la simulación de dos campos de Lieb-Liniger acoplados. Se ha comprobado que variando la intensidad del acoplo se produce una transición de fase en la que uno de los campos queda despoblado, y que dicha transición deja signaturas en la energía, la densidad de partículas y las funciones de correlación.

Los resultados obtenidos en este trabajo permitirán la realización experimental mediante simuladores cuánticos de sistemas análogos a los modelos teóricos empleados, en los que se espera reproducir la transición de fase. Dichos experimentos se realizarán en una colaboración con el grupo de Andreas Wallraf (ETH-Zürich).

Bibliografía

[1] A. Altland y B. Simons, *Condensed Matter Field Theory.* Cambridge University Press, 2010.

[2] T. Giamarchi, *Quantum Physics in One Dimension.* Clarendon Press, 2003.

[3] F. Verstraete y J. I. Cirac, *Continuous matrix product states for quantum fields,* *Physical Review Letters* **104** no. 19, (2010) 1–4, arXiv:1002.1824.

[4] F. Quijandría, J. J. García-Ripoll, y D. Zueco, *Continuous matrix product states for coupled fields: Application to Luttinger Liquids and quantum simulators,* *Physical Review B - Condensed Matter and Materials Physics* **90** no. 23, (2014) 1–10, arXiv:1409.4709v1.

[5] C. Cohen-Tannoudji, B. Diu, y F. Laloë, *Quantum Mechanics. Volume one.* Hermann-John Wiley & Sons, 1977.

[6] M. A. Nielsen y I. L. Chuang, *Quantum Computation and Quantum Information.* Cambridge University Press, 2000.

[7] A. Rivas y S. F. Huelga, *Open Quantum Systems. An Introduction.* Springer, 2011. arXiv:1104.5242v2.

[8] J. Eisert, *Entanglement and tensor network states, Modeling and Simulation* **3** no. 520, (2013) 39, arXiv:1308.3318.

[9] C. Cohen-Tannoudji, J. Dupont-Roc, y G. Grynberg, *Photons and Atoms: Introduction to Quantum Electrodynamics.* Wiley, 1989.

[10] C. Cohen-Tannoudji, B. Diu, y F. Laloë, *Quantum Mechanics. Volume two.* Hermann-John Wiley & Sons, 1977.

[11] M. Rispler, *Continuous Matrix Product State Representations for Quantum Field Theory.* Msc dissertation, Imperial College London, 2012. https://workspace.imperial.ac.uk/theoreticalphysics/Public/MSc/Dissertations/2012/ManuelRisplerDissertation.pdf.

[12] R. Orús, *A practical introduction to tensor networks: Matrix product states and projected entangled pair states,* *Annals of Physics* **349** (2014) 117–158, arXiv:1306.2164.

[13] J. Haegeman, J. I. Cirac, T. J. Osborne, y F. Verstraete, *Calculus of continuous matrix product states,* *Physical Review B - Condensed Matter and Materials Physics* **88** no. 8, (2013) 1–47, arXiv:1211.3935.

[14] M. Galassi, *GNU Scientific Library Reference Manual,* 2009. http://www.gnu.org/software/gsl/.

[15] S. G. Johnson, *The NLopt nonlinear-optimization package,* 2014. http://ab-initio.mit.edu/wiki/index.php/NLopt.

[16] M. Olshanii y V. Dunjko, *Short-distance correlation properties of the Lieb-Liniger system and momentum distributions of trapped one-dimensional atomic gases.,* *Physical Review Letters* **91** no. 9, (2003) 090401, arXiv:cond-mat/0210629.

[17] M. Zvonarev, *Correlations in 1D boson and fermion systems: exact results.* PhD thesis, Copenhagen University, 2005.

[18] H. Moritz, T. Stöferle, M. Köhl, y T. Esslinger, *Exciting Collective Oscillations in a Trapped 1D Gas*, *Physical Review Letters* **91** no. 25, (Dec., 2003) , arXiv:cond-mat/0307607.

[19] M. A. Cazalilla, *Bosonizing one-dimensional cold atomic gases*, *Journal of Physics B: Atomic, Molecular and Optical Physics* **37** no. 7, (Apr., 2004) S1–S47, arXiv:cond-mat/0307033.

[20] A. K. Kolezhuk, *Stability of low-dimensional multicomponent dilute Bose gases*, *Physical Review A - Atomic, Molecular, and Optical Physics* **81** no. 1, (2010) 1–6, arXiv:0903.1647v4.

[21] S. B. Papp, J. M. Pino, y C. E. Wieman, *Tunable miscibility in a dual-species Bose-Einstein condensate*, *Physical Review Letters* **101** no. 4, (2008) 1–5, arXiv:0802.2591.

[22] L. E. Sadler, J. M. Higbie, S. R. Leslie, M. Vengalattore, y D. M. Stamper-Kurn, *Spontaneous symmetry breaking in a quenched ferromagnetic spinor Bose-Einstein condensate.*, *Nature* **443** no. 7109, (2006) 312–315, arXiv:cond-mat/0605351.

[23] E. Timmermans, *Phase separation of Bose-Einstein condensates*, *Physical Review Letters* **81** no. 26, (1997) 11, arXiv:cond-mat/9709301.

A Cálculos con cMPS

Para describir la dinámica en el formalismo cMPS se utilizará un espacio de Hilbert auxiliar $\mathcal{H}_{aux}$ además del espacio del campo $\mathcal{H}_f$, $\mathcal{H} = \mathcal{H}_{aux} \otimes \mathcal{H}_f$. El operador hamiltoniano en este espacio es

$$H = Q \otimes \mathbb{1} + R \otimes \psi^\dagger \tag{A.1}$$

El estado cMPS se define como la acción del operador U sobre el vacío, trazando sobre los grados de libertad del sistema auxiliar:

$$|\chi\rangle = \mathrm{tr}_{aux}\left[\mathcal{P}\exp\int_0^L H\mathrm{d}s\right]|\Omega\rangle \tag{A.2}$$

En el hamiltoniano usado, se puede asociar la parte de $H_0 = Q \otimes I$ como el hamiltoniano libre y $V = R \otimes \psi^\dagger$ como una interacción. Trabajando en la imagen de interacción:

$$U_I(x_0, x) = U_0^\dagger(x_0, x)U(x_0, x) \tag{A.3}$$

Haciendo $M_Q(a, b) = \mathrm{e}^{H_0(b-a)}$ y $V(x) = R(x) \otimes \psi^\dagger(x)$, podemos escribir el estado cMPS como

$$\begin{aligned}|\chi\rangle &= \mathrm{tr}_{aux}\left[\sum_{n=0}^{\infty}\int_{0\le x_1\le\cdots\le L} M_Q(0, x_1)(R(x_1) \otimes \psi^\dagger(x_1))M_Q(x_1, x_2)\cdots M_Q(x_n, L)\right]|\Omega\rangle = \\ &= \mathrm{tr}_{aux}\left[\sum_{n=0}^{\infty}\int M_Q(0, x_1)R(x_1)M_Q(x_1, x_2)\cdots R(x_n)M_Q(x_n, L)\right]\psi^\dagger(x_1)\cdots\psi^\dagger(x_n)|\Omega\rangle\end{aligned} \tag{A.4}$$

Por lo tanto, el estado cMPS se puede expresar como una superposición de estados con $1, 2, \ldots, n$ partículas (estados de Fock):

$$|\chi\rangle = \sum_n \int_{0<x_1<\cdots<x_n<L} \mathrm{d}x_1\cdots\mathrm{d}x_n\phi(x_1, \cdots, x_n)\psi^\dagger(x_1)\cdots\psi^\dagger(x_n)|\Omega\rangle \tag{A.5}$$

donde los coeficientes de la combinación lineal son

$$\phi(x_1, \cdots, x_n) = \langle\Omega|\,\psi(x_1)\cdots\psi(x_n)\,|\chi\rangle \tag{A.6}$$

Teniendo en cuenta las relaciones de conmutación para los operadores del campo

$$\begin{aligned}&[\psi(x), \psi^\dagger(x_1)] = \delta(x - x_1) \qquad [\psi(x), \psi^\dagger(x_1)\psi^\dagger(x_2)] = \psi^\dagger(x_2)\delta(x - x_1) + \psi^\dagger(x_1)\delta(x - x_2) \\ &[\psi(x), \psi^\dagger(x_1)\cdots\psi^\dagger(x_n)] = \sum_m \psi^\dagger(x_1)\cdots\psi^\dagger(x_{m-1})\psi^\dagger(x_{m+1})\cdots\psi^\dagger(x_n)\delta(x - x_m)\end{aligned} \tag{A.7}$$

con lo cual, utilizando el desarrollo de la exponencial que hemos encontrado antes e integrando en x_m, se obtiene

$$\left[\psi(x), \mathcal{P}\exp\int_0^L H\mathrm{d}s\right] = \left(\mathcal{P}\exp\int_0^x H\mathrm{d}s\right)R(x)\left(\mathcal{P}\exp\int_x^L H\mathrm{d}s\right) \tag{A.8}$$

Una vez conocido este conmutador, ya podemos expresar el estado cMPS desarrollado en los estados de Fock (1.24), pasando los operadores de destrucción a la parte derecha (*normal ordering*), teniendo en cuenta que:

$$\begin{aligned}\psi(x)\mathcal{P}\exp\int_0^L H\mathrm{d}s\,|\Omega\rangle &= \mathcal{P}\exp\int_0^L H\mathrm{d}s\psi\,|\Omega\rangle + \left(\mathcal{P}\exp\int_0^x H\mathrm{d}s\right)R(x)\left(\mathcal{P}\exp\int_x^L H\mathrm{d}s\right)|\Omega\rangle \\ &= \left(\mathcal{P}\exp\int_0^x H\mathrm{d}s\right)R(x)\left(\mathcal{P}\exp\int_x^L H\mathrm{d}s\right)|\Omega\rangle\end{aligned} \tag{A.9}$$

Aplicándolo sucesivamente con cada uno de los ψ:

$$\begin{aligned}\phi(x_1,\cdots,x_n) &= \langle\Omega|\,\psi(x_1)\cdots\psi(x_n)\,|\chi\rangle = \langle\Omega|\,\psi(x_1)\cdots\psi(x_n)\mathrm{tr}_{aux}\left[\mathcal{P}\exp\int_0^L H\mathrm{d}s\right]|\Omega\rangle \\ &= \langle\Omega|\,\psi(x_1)\cdots\mathrm{tr}_{aux}\left[\left(\mathcal{P}\exp\int_0^{x_n} H\mathrm{d}s\right)R(x_n)\left(\mathcal{P}\exp\int_{x_n}^L H\mathrm{d}s\right)\right]|\Omega\rangle \\ &= \langle\Omega|\,\mathrm{tr}_{aux}\left[\left(\mathcal{P}\mathrm{e}^{\int_0^{x_1}H\mathrm{d}s}\right)R(x_1)\left(\mathcal{P}\mathrm{e}^{\int_{x_1}^{x_2}H\mathrm{d}s}\right)\cdots\left(\mathcal{P}\mathrm{e}^{\int_{x_{n-1}}^{x_n}H\mathrm{d}s}\right)R(x_n)\left(\mathcal{P}\mathrm{e}^{\int_{x_n}^{L}H\mathrm{d}s}\right)\right]|\Omega\rangle \\ &= \mathrm{tr}_{aux}[M_Q(0,x_1)R(x_1)M_Q(x_1,x_2)\cdots R(x_n)M_Q(x_n,L)] \qquad \text{(A.10)}\end{aligned}$$

A.1. Normalización

Una vez obtenido el *ket* del estado cMPS, su *bra* asociado es simplemente

$$\langle\chi| = \sum_{n=0}^{\infty}\int \mathrm{d}y_1\cdots\mathrm{d}y_n\,\langle\Omega|\,\psi(y_1)\cdots\psi(y_n)\mathrm{tr}_{aux}[M_Q^\star(0,y_1)R^\star(y_1)M_Q^\star(y_1,y_2)\cdots R^\star(y_n)M_Q^\star(y_n,L)] \qquad \text{(A.11)}$$

por lo que su norma es:

$$\begin{aligned}\langle\chi|\chi\rangle = \sum_{n,m=0}^{\infty}\int \mathrm{d}y_1\cdots\mathrm{d}y_m\int \mathrm{d}x_1\cdots\mathrm{d}x_n\,\langle\Omega|\,\psi(y_1)\cdots\psi(y_m)\psi^\dagger(x_1)\psi^\dagger(x_n)\,|\Omega\rangle\times \\ \times\,\mathrm{tr}_{aux}[M_Q(0,x_1)R(x_1)\cdots R(x_n)M_Q(x_n,L)]\mathrm{tr}_{aux}[M_Q^\star(0,y_1)R^\star(y_1)\cdots R^\star(y_m)M_Q^\star(y_m,L)] \qquad \text{(A.12)}\end{aligned}$$

La parte que depende de los campos es simplemente el producto escalar de dos estados del espacio de Fock, que forman parte de una base ortonormal:

$$\langle\Omega|\,\psi(y_1)\cdots\psi(y_m)\psi^\dagger(x_1)\psi^\dagger(x_n)\,|\Omega\rangle = \delta_{nm}\delta(x_1-y_1)\cdots\delta(x_n-y_n) \qquad \text{(A.13)}$$

La parte del sistema auxiliar se puede simplificar teniendo en cuenta las propiedades de las trazas y exponenciales de los productos tensoriales:

$$\begin{aligned}\mathrm{tr}A\mathrm{tr}B = \mathrm{tr}(A\otimes B) \qquad \mathrm{tr}AC\mathrm{tr}BD = \mathrm{tr}AC\otimes BD = \mathrm{tr}(A\otimes B)(C\otimes D) \\ \exp A\otimes\exp B = \exp(A\otimes\mathbb{1}+\mathbb{1}\otimes B) \qquad \text{(A.14)}\end{aligned}$$

con lo cual,

$$\begin{aligned}\langle\chi|\chi\rangle &= \sum_{n=0}^{\infty}\int \mathrm{d}x_1\cdots\mathrm{d}x_n\mathrm{tr}_{aux}[M_Q(0,x_1)R(x_1)\cdots R(x_n)M_Q(x_n,L)] \qquad \text{(A.15)} \\ &\quad\times\mathrm{tr}_{aux}[M_Q^\star(0,x_1)R^\star(x_1)\cdots R^\star(x_m)M_Q^\star(x_m,L)] \\ &= \sum_{n=0}^{\infty}\int \mathrm{d}x_1\cdots\mathrm{d}x_n\mathrm{tr}_{aux}[M_Q(0,x_1)\otimes M_Q^\star(0,x_1)R(x_1)\otimes R^\star(x_1)\cdots M_Q(x_n,L)\otimes M_Q^\star(x_n,L)] \\ &= \sum_{n=0}^{\infty}\int \mathrm{d}x_1\cdots\mathrm{d}x_n\mathrm{tr}_{aux}\left[\left(\exp\int_0^{x_1}\mathrm{d}s(Q(s)\otimes\mathbb{1}+\mathbb{1}\otimes Q^\star(s)\right)R(x_1)\otimes R^\star(x_1)\times\right. \\ &\quad\left.\times\left(\exp\int_{x_1}^{x_2}\mathrm{d}s(Q(s)\otimes\mathbb{1}+\mathbb{1}\otimes Q^\star(s)\right)\cdots R(x_n)\otimes R^\star(x_n)\left(\exp\int_{x_n}^{L}\mathrm{d}s(Q(s)\otimes\mathbb{1}+\mathbb{1}\otimes Q^\star(s)\right)\right]\end{aligned}$$

Recordando el desarrollo de la exponencial (1.15), finalmente tenemos:

$$\langle\chi|\chi\rangle = \mathcal{P}\exp\int_0^L \mathrm{d}s(Q(s)\otimes\mathbb{1}+\mathbb{1}\otimes Q^\star(s)+R(s)\otimes R^\star(s)) = \mathcal{P}\exp\int_0^L T(s)\mathrm{d}s \qquad \text{(A.16)}$$

donde se ha definido el operador de transferencia T como:

$$T = Q\otimes\mathbb{1}+\mathbb{1}\otimes Q^\star+R^\star R \qquad \text{(A.17)}$$

A.2. Funciones de correlación campo-campo

En primer lugar nos interesará calcular la siguiente función de correlación:

$$C(x,y) = \langle\chi|\,\psi^\dagger(x)\psi(y)\,|\chi\rangle \qquad\qquad y > x \tag{A.18}$$

Para hacerlo calcularemos $\psi(y)\,|\chi\rangle$ y $\langle\chi|\,\psi^\dagger(x)$ y realizaremos su producto escalar.

$$\psi(y)\,|\chi\rangle = \mathrm{tr}_{aux}\left[\psi(y)\mathcal{P}\exp\int_0^L H(s)\mathrm{d}s\right]|\Omega\rangle \tag{A.19}$$

que aplicando la relación (A.9) queda

$$= \mathrm{tr}_{aux}\left[\left(\mathcal{P}\exp\int_0^L H\mathrm{d}s\right)R(y)\left(\mathcal{P}\exp\int_0^L H\mathrm{d}s\right)\right]|\Omega\rangle \tag{A.20}$$

La primera integral la separamos en dos, una entre 0 y x y otra entre x e y. Las tres integrales las podemos calcular con el desarrollo de Dyson (1.15):

$$\begin{aligned}\psi(y)\,|\chi\rangle =&\mathrm{tr}_{aux}\left[\sum_{m_1,m_2,m_3}\int \mathrm{d}x_1\mathrm{d}x_{m_1}\int \mathrm{d}y_1 y_{m_2}\int \mathrm{d}z_1\mathrm{d}z_{m_3}M_Q(0,x_1)R(x_1)\cdots M_Q(x_{m_1},x)\mathbb{1}M_Q(x,y_1)\right.\\ &\left.\times R(y_1)\cdots M_Q(y_{m_2},y)R(y)M_Q(y,z_1)R(z_1)\cdots M_Q(z_{m_3},L)\right]\psi^\dagger(x_1)\cdots\psi^\dagger(z_{m_3})\,|\Omega\rangle\end{aligned} \tag{A.21}$$

Procedemos del mismo modo con $\langle\chi|\,\psi^\dagger(x)$, teniendo en cuenta que ahora la integral que debemos separar en dos va entre x y L:

$$\begin{aligned}\langle\chi|\,\psi^\dagger(x) =&\mathrm{tr}_{aux}\left[\sum_{m_1,m_2,m_3}\int \mathrm{d}x_1\cdots\mathrm{d}x_{m_1}\int \mathrm{d}y_1\cdots y_{m_2}\int \mathrm{d}z_1\cdots\mathrm{d}z_{m_3}\,\langle\Omega|\,\psi(x_1)\cdots\psi(z_{m_3})M_Q^\star(0,x_1)\right.\\ &\left.\times R^\star(x_1)\cdots M_Q^\star(x_{m_1},x)R^\star(x)M_Q^\star(x,y_1)R^\star(y_1)\cdots M_Q^\star(y_{m_2},y)IM_Q^\star(y,z_1)R^\star(z_1)\cdots M_Q^\star(z_{m_3},L)\right]\end{aligned} \tag{A.22}$$

Tal y como pasaba al calcular la normalización, al hacer el producto escalar, la parte que depende de los campos se reduce a un producto de deltas que expresa la ortonormalidad. Y para la parte del sistema auxiliar, de nuevo podemos operar con las propiedades del producto tensorial:

$$\begin{aligned}\langle\chi|\,\psi^\dagger(x)\psi(y)\,|\chi\rangle =&\mathrm{tr}_{aux}\big[M_Q(0,x_1)\otimes M_Q^\star(0,x_1)R(x_1)\otimes R^\star(x_1)\cdots M_Q(x_{m_1},x)\otimes M_Q^\star(x_{m_1},x)\mathbb{1}\otimes R^\star(x)\\ &\times M_Q(x,y_1)\otimes M_Q^\star(x,y_1)R(y_1)\otimes R^\star(y_1)\cdots M_Q(y_{m_2},y)\otimes M_Q^\star(y_{m_2},y)R(y)\otimes\mathbb{1}\\ &\times M_Q(y,z_1)\otimes M_Q^\star(y,z_1)R(z_1)\otimes R^\star(z_1)\cdots M_Q(z_{m_3},L)\otimes M_Q^\star(z_{m_3},L)\big]\end{aligned} \tag{A.23}$$

volviendo a aplicar la serie de Dyson, y con la matriz de transferencia $T(s)$, se consigue el siguiente resultado:

$$\langle\chi|\,\psi^\dagger(x)\psi(y)\,|\chi\rangle = \mathrm{tr}_{aux}\left[\left(\mathcal{P}\mathrm{e}^{\int_0^x T(s)\mathrm{d}s}\right)\mathbb{1}\otimes R^\star(x)\left(\mathcal{P}\mathrm{e}^{\int_x^y T(s)\mathrm{d}s}\right)R(y)\otimes\mathbb{1}\left(\mathcal{P}\mathrm{e}^{\int_y^L T(s)\mathrm{d}s}\right)\right] \tag{A.24}$$

A.3. Funciones de correlación densidad-densidad

La correlación entre la densidad de partículas en dos puntos $y < x$, es proporcional a la siguiente correlación:

$$\langle n(x)n(y)\rangle \sim \langle\chi|\,\psi^\dagger(x)\psi^\dagger(y)\psi(y)\psi(x)\,|\chi\rangle \tag{A.25}$$

Para calcularla, debemos conocer el valor de $\psi(y)\psi(x)\,|\chi\rangle$:

$$\begin{aligned}
\psi(y)\psi(x)\,|\chi\rangle =&\psi(y)\mathrm{tr}_{aux}\left[\psi(x)\mathcal{P}\exp\int_0^L H\mathrm{d}s\right]|\Omega\rangle = \psi(y)\mathrm{tr}_{aux}\left[\left(\mathcal{P}\exp\int_0^x H\mathrm{d}s\right)R(x)\left(\mathcal{P}\exp\int_x^L H\mathrm{d}s\right)\right]\\
=&\mathrm{tr}_{aux}\left[\left(\mathcal{P}\exp\int_0^y H\mathrm{d}s\right)R(y)\left(\mathcal{P}\exp\int_y^x H\mathrm{d}s\right)R(x)\left(\mathcal{P}\exp\int_x^L H\mathrm{d}s\right)\right]|\Omega\rangle\\
=&\sum_{m_1,m_2,m_3}\int \mathrm{d}x_1\cdots\mathrm{d}x_{m_1}\int \mathrm{d}y_1\cdots\mathrm{d}y_{m_2}\int \mathrm{d}z_1\cdots z_{m_3}\mathrm{tr}_{aux}[M_Q(0,x_1)R(x_1)\cdots R(x_{m_1})M_Q(x_{m_1},y)R(y)\\
&\times M_Q(y,y_1)R(y_1)\cdots R(y_{m_2})M_Q(y_{m_2},x)R(x)M_Q(x,z_1)R(z_1)\cdots R(z_{m_3})M_Q(z_{m_3})]\psi^\dagger(x_1)\cdots\psi^\dagger(z_{m_3})\,|\Omega\rangle
\end{aligned} \tag{A.26}$$

Para obtener el correspondiente *bra* se procede de modo análogo. Como ocurría en los casos anteriores, al realizar el producto escalar la parte que depende de los campos se reduce a la condición de ortonormalidad, y la parte del sistema auxiliar se *simplifica* al expresarla en términos de productos tensoriales:

$$\begin{aligned}
\langle\psi^\dagger(x)\psi^\dagger(y)\psi(y)\psi(x)\rangle =&\mathrm{tr}_{aux}\big[M_Q(0,x_1)\otimes M_Q^\star(0,x_1)R(x_1)\otimes R^\star(x_1)\cdots R(y)\otimes R^\star(y)M_Q(y,y_1)\otimes M_Q^\star(y,y_1)\\
&\times R(y_1)\otimes R^\star(y_1)\cdots M_Q(y_{m_2},x)\otimes M_Q^\star(y_{m_2},x)R(x)\otimes R^\star(x)M_Q(x,z_1)\otimes M_Q^\star(x,z_1)\\
&\times R(z_1)\otimes R^\star(z_1)\cdots R(z_{m_3})\otimes R(z_{m_3})M_Q(z_{m_3},L)\otimes M_Q^\star(z_{m_3},L)\big]
\end{aligned} \tag{A.27}$$

que haciendo uso del desarrollo de Dyson, resulta

$$\langle n(x)n(y)\rangle = \mathrm{tr}_{aux}\left[\left(\mathcal{P}\mathrm{e}^{\int_0^y T(s)}\mathrm{d}s\right)R(y)\otimes R^\star(y)\left(\mathcal{P}\mathrm{e}^{\int_y^x T(s)}\mathrm{d}s\right)R(x)\otimes R^\star(x)\left(\mathcal{P}\mathrm{e}^{\int_x^L T(s)}\mathrm{d}s\right)\right] \tag{A.28}$$

B Código de los programas

B.1. Modelo Lieb-Liniger en un campo

```
/**
Program to find the minimal energy in the Lieb-Liniger model as a function of the
    interaction parameter c in the framework of the Continuous Matrix Product
    State (cMPS)
Based on the code by M. Rispler
Jorge Alda
*/

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
//#include <string.h>

//GNU Scientific Library headers
#include <gsl/gsl_math.h>
#include <gsl/gsl_rng.h>//Random number generator
#include <gsl/gsl_vector.h> //Vector handling
#include <gsl/gsl_matrix.h> //Matrix handling
#include <gsl/gsl_complex.h> //Complex numbers
#include <gsl/gsl_complex_math.h> //Complex number operations
#include <gsl/gsl_blas.h> //Linear algebra engine
#include <gsl/gsl_linalg.h>

//Optimization
#include <nlopt.h>

//Algorithm to find the steady state of the Liouvillian
#define QR

void extract(const gsl_vector *x, gsl_matrix_complex *H0, gsl_matrix_complex *R0,
    unsigned int D);
void enlarge(double *orig, double *dest, unsigned int n, gsl_rng *rng);
void spre(gsl_matrix_complex *A,  gsl_matrix_complex *preA, unsigned int D);
void reprM(gsl_matrix_complex *R0, unsigned int dim);
void spost(const gsl_matrix_complex *A, gsl_matrix_complex *postA, unsigned int
    trans, unsigned int D);
void steadystatefinder(gsl_matrix_complex *H, gsl_matrix_complex *R,
    gsl_matrix_complex *rho, unsigned int D);
double matrixnorm(gsl_matrix_complex *A);
void powermethod (gsl_matrix_complex *A, gsl_vector_complex *v, unsigned int D);
int gsl_linalg_complex_QR_decomp (gsl_matrix_complex * A);
double particlenumber(gsl_matrix_complex *R, gsl_matrix_complex *rho, unsigned
    int D);
double energydensity(gsl_matrix_complex *H, gsl_matrix_complex *R,
    gsl_matrix_complex *rho, double c, unsigned int D);
void save(double *x, char *t, unsigned int D);

double barrier(unsigned int n, const double *x, const double *grad, void
    *myfuncdata);
double enr, np, eps=1e-6, Npart=0.53;
```

```
int main(){
    int D, j, k, Nbuc;
    double min, max, delta;
    char input[200], output[500];
    FILE *settings;
    if((settings=fopen("settings.txt", "rt"))==NULL){
        printf("no existe archivo settings");
        return -1;
    }
    fscanf(settings, "%s", input);
    fscanf(settings, "%s", output);
    fscanf(settings, "%d", &D);
    fscanf(settings, "%lf", &min);
    fscanf(settings, "%lf", &max);
    fscanf(settings, "%lf", &delta);
    fscanf(settings, "%d", &Nbuc);
    double x[2*D*D];
    FILE *f;
    time_t start, stop;


    if (strcmp(input, "r")==0){
        printf("random\n");
    gsl_rng *rng = gsl_rng_alloc(gsl_rng_taus);
    for(j=0; j<2*(D)*(D); j++){
       x[j] = gsl_rng_uniform(rng);
    }
    }else{
    f = fopen(input, "rt");
    for(j=0; j<2*(D)*(D); j++){
       fscanf(f, "%lf", &x[j]);
    }
    fclose(f);
    printf("leido el archivo %s", input);
    }

    double c[2], mm;
    c[1] = D;
    c[0] = min;


    nlopt_opt opt = nlopt_create(NLOPT_LN_SBPLX, 2*D*D);
    nlopt_set_min_objective(opt, barrier, c);
    nlopt_set_xtol_rel(opt, 1e-4);
    nlopt_set_maxeval(opt, 30000);

    //Find the groundstate
    sprintf(arch, "LL_%d.txt", D);
    time(&start);
    for(j=0; j<Nbuc; j++){
    for(c[0] = min; c[0]<max; c[0]+= delta){
        nlopt_optimize(opt, x, &mm);
        f = fopen(arch, "at");
        fprintf(f, "%lf\t%lf\n", c[0], enr);
        fclose(f);
```

```
    }
    printf("%lf\n", enr);
    for(; c[0]>min; c[0]-= delta){
        nlopt_optimize(opt, x, &mm);
        f = fopen(arch, "at");
        fprintf(f, "%lf\t%lf\n", c[0], enr);
        fclose(f);
    }
    }
    time(&stop):
    save(x, output, D);
    printf("%0.f s\n", difftime(stop, start));
    return 0;
}


double barrier(unsigned int n, const double *x, const double *grad, void
    *myfuncdata){
    //Barrier function to be minimized
    double e, N, L;
    double *c = (double *)myfuncdata;
    unsigned int D = (unsigned int)c[1];
    gsl_vector_view v = gsl_vector_view_array(x, 2*D*D);
    gsl_matrix_complex *H = gsl_matrix_complex_alloc(D, D);
    gsl_matrix_complex *R = gsl_matrix_complex_alloc(D, D);
    gsl_matrix_complex *rho = gsl_matrix_complex_alloc(D, D);
    extract(&v.vector, H, R, D);
    steadystatefinder(H, R, rho, D);
    e = energydensity(H, R, rho, c[0], D);
    enr = e;
    N = particlenumber(R, rho, D);
    np = N;
    gsl_matrix_complex_free(H);
    gsl_matrix_complex_free(R);
    gsl_matrix_complex_free(rho);
    L = e - 0.0001*log((N-Npart)); //We're using a logarithmic barrier
    return L;
}

void reprM(gsl_matrix_complex *R0, unsigned int dim){
    // Auxiliar function to print matrices
    unsigned int m, n;
    for(m=0; m<dim; m++){
        for(n=0; n<dim; n++){
            printf("%lf+%lfi\t", GSL_REAL(gsl_matrix_complex_get(R0, m, n)),
                GSL_IMAG(gsl_matrix_complex_get(R0, m, n)));
        }
        printf( "\n");
    }
    printf( "\n");
}

void reprv(gsl_vector_complex *v){
    // Auxiliar function to print vectors
    unsigned int D = v->size, i;
```

```
    for(i=0; i<D; i++){
        printf("%.3lf+%.3lfi\t", GSL_REAL(gsl_vector_complex_get(v, i)),
            GSL_IMAG(gsl_vector_complex_get(v, i)));
    }
    printf("\n");
}

void save(double *x, char *t, unsigned int D){
    // Auxiliar function to save the state to file
    int i, j;
    char t1[500], t2[500];
    sprintf(t1, "KR_%s.txt", t);
    sprintf(t2, "rho_%s.txt", t);
    FILE *f = fopen(t1, "wt");
    for(i=0; i<2*D*D; i++){
        fprintf(f, "%lf\n", x[i]);
    }
    fclose(f);
    gsl_vector_view v = gsl_vector_view_array(x, 2*D*D);
    gsl_matrix_complex *H = gsl_matrix_complex_alloc(D, D);
    gsl_matrix_complex *R = gsl_matrix_complex_alloc(D, D);
    gsl_matrix_complex *rho = gsl_matrix_complex_alloc(D, D);
    extract(&v.vector, H, R, D);
    steadystatefinder(H, R, rho, D);
    f = fopen(t2, "wt");
    for(j=0; j<D; j++){
        for(i=0; i<D; i++){
            fprintf(f, "%lf\t%lf_i\n", GSL_REAL(gsl_matrix_complex_get(rho, i,
                j)), GSL_IMAG(gsl_matrix_complex_get(rho, i, j)));
        }
    }
    fclose(f);
    gsl_matrix_complex_free(H);
    gsl_matrix_complex_free(R);
    gsl_matrix_complex_free(rho);
}


void extract(const gsl_vector *x, gsl_matrix_complex *H0, gsl_matrix_complex *R0,
    unsigned int D){
    // Unpacks H and R from a compact vector storage
    unsigned int a,n,m;
    for(m=0; m<D;m++){
        gsl_matrix_complex_set(H0, m, m, gsl_complex_rect(gsl_vector_get(x,m),0));
    }
    a = D;
    for(m=0; m<D;m++){
        for(n=m+1; n<D;n++){
            gsl_matrix_complex_set(H0, m, n,
                gsl_complex_rect(gsl_vector_get(x,a), gsl_vector_get(x,a+1)));
            gsl_matrix_complex_set(H0, n, m,
                gsl_complex_rect(gsl_vector_get(x,a), -gsl_vector_get(x,a+1)));
            a += 2;
        }
    }
```

```
    a=0;
    for(m=0;m<D;m++){
        for(n=0;n<D;n++){
            gsl_matrix_complex_set(R0, n, m,
                gsl_complex_rect(gsl_vector_get(x,a+D*D),0));
            a++;
        }
    }
}

void enlarge(double *orig, double *dest, unsigned int n, gsl_rng *rng){
    // Creates a D+1 dimensional state form a D dimensional one
    unsigned int i, j, k, l;
    double alfa =0;
    for(i=0; i<n; i++){
        dest[i] = orig[i];
    }
    l=i;
    dest[i] = alfa*gsl_rng_uniform(rng);
    i++;
    for(j=0; j<n-1; j++){
        for(k=0; k<2*(n-j-1); k++){
            i++;
            l++;
            dest[l] = orig[i-2];

        }
        dest[l+2] = alfa*gsl_rng_uniform(rng);
        dest[l+1] = alfa*gsl_rng_uniform(rng);
        l+=2;
    }
    dest[l+2] = alfa*gsl_rng_uniform(rng);
    dest[l+1] = alfa*gsl_rng_uniform(rng);

    for(i=0; i<(n+1)*(n+1); i++){
        j = i/(n+1);
        k = i%(n+1);
        if((j<n)&&(k<n)){
            dest[i+(n+1)*(n+1)] = orig[n*j+k+n*n];
        }else{
            dest[i+(n+1)*(n+1)] = alfa*gsl_rng_uniform(rng);
        }
    }

}

void spre(gsl_matrix_complex *A, gsl_matrix_complex *preA, unsigned int D){
    // Suoeroperator for left matrix multiplication
    gsl_matrix_complex_set_zero(preA);
    unsigned int i;
    for(i=0; i<D; i++){
        gsl_matrix_complex_view block =
            gsl_matrix_complex_submatrix(preA, i*D, i*D,D,D);
        gsl_matrix_complex_memcpy(&block.matrix,A);
```

```
    }
}

void spost(const gsl_matrix_complex *A, gsl_matrix_complex *postA, unsigned int
    trans, unsigned int D){
    // Superoperator for right matrix multiplication
    gsl_matrix_complex_set_zero(postA);
    unsigned int i, n, m;
    gsl_complex d;
    for(m=0;m<D;m++){
        for(n=0;n<D;n++){
            if(trans){
                d = gsl_complex_conjugate(gsl_matrix_complex_get(A, n, m));
            }else{
                d = gsl_matrix_complex_get(A, m, n);
            }
            for(i=0;i<D;i++){
                gsl_matrix_complex_set(postA,i+D*n, i+D*m, d);
            }
        }
    }
}

void steadystatefinder(gsl_matrix_complex *H, gsl_matrix_complex *R,
    gsl_matrix_complex *rho, unsigned int D){
    // Finds the steady state density matrix
    int i=0, m, n;
    gsl_complex tr = GSL_COMPLEX_ZERO;
    gsl_rng *rng = gsl_rng_alloc(gsl_rng_taus);
    gsl_matrix_complex *L = gsl_matrix_complex_alloc(D*D, D*D);
    gsl_matrix_complex *post = gsl_matrix_complex_alloc(D*D, D*D);
    gsl_matrix_complex *pre = gsl_matrix_complex_alloc(D*D, D*D);
    gsl_matrix_complex *RtR = gsl_matrix_complex_alloc(D, D);
    gsl_vector_complex *vectr = gsl_vector_complex_alloc(D*D);
    spre(H,L, D);
    spost(H, post, 0, D);
    gsl_matrix_complex_sub(L,post);
    spre(R, pre, D);
    spost(R, post, 1, D);
    gsl_blas_zgemm(CblasNoTrans, CblasNoTrans, gsl_complex_rect(1,0), pre,post,
        gsl_complex_rect(0, -1), L);
    gsl_blas_zgemm(CblasConjTrans, CblasNoTrans, gsl_complex_rect(1,0), R, R,
        gsl_complex_rect(0,0), RtR);
    spre(RtR, pre, D);
    spost(RtR, post, 0, D);
    gsl_matrix_complex_add(pre, post);
    gsl_matrix_complex_scale(pre, gsl_complex_rect(0.5,0));
    gsl_matrix_complex_sub(L, pre);

    gsl_matrix_complex_free(pre);
    gsl_matrix_complex_free(RtR);
    gsl_matrix_complex_free(post);

    #ifdef QR
```

```
    gsl_linalg_complex_QR_decomp(L);
    for(i=D*D-1; i>=0; i--){
        if(gsl_complex_abs(gsl_matrix_complex_get(L, i, i)) < eps){
            gsl_vector_complex_set(vectr, i, gsl_complex_rect(0,0));
        }else{
            gsl_complex hh = gsl_complex_rect(0,0);
            int j;
            for(j=i-1; j<D*D; j++){
                hh = gsl_complex_sub(hh,
                    gsl_complex_mul(gsl_matrix_complex_get(L,
                    i,j)*gsl_vector_complex_get(vectr, j) ));
            }
            gsl_vector_complex_set(vectr, i, gsl_complex_div(hh,
                gsl_matrix_complex_get(L, i,i) ));
        }
    }
    #else
    for(i=0; i<vectr->size; i++){
        gsl_vector_complex_set(vectr, i,
            gsl_complex_rect(gsl_rng_uniform(rng),gsl_rng_uniform(rng)));
    }
    gsl_vector_complex_set(vectr, vectr->size-1, GSL_COMPLEX_ZERO);
    powermethod(L, vectr, D*D);
    gsl_rng_free(rng);
    #endif // QR

    i=0;
    for(m=0; m<D; m++){
        for(n=0; n<D; n++){
            gsl_matrix_complex_set(rho, n, m, gsl_vector_complex_get(vectr, i));
            if (m==n) tr= gsl_complex_add(tr, gsl_vector_complex_get(vectr, i));
            i++;
        }
    }
    gsl_matrix_complex_scale(rho, gsl_complex_inverse(tr));
    gsl_matrix_complex_free(L);
    gsl_vector_complex_free(vectr);
}

double matrixnorm(gsl_matrix_complex *A){
    /*Compute the inf-norm of matrix A*/
    const size_t M = A->size1;
    double n = 0, t;
    int i, j;
    for(i=0; i<M; i++){
        for(j=0; j<M; j++){
            t = gsl_complex_abs(gsl_matrix_complex_get(A, i, j));
            if(t > n) n = t;
        }
    }
    return n;
}

double vectornorm(gsl_vector_complex *v){
```

```
    /*Compute the inf-norm of vector v*/
    const size_t M = v->size;
    double n = 0, t;
    int i;
    for(i=0; i<M; i++){
            t = gsl_complex_abs(gsl_vector_complex_get(v, i));
            if(t > n) n = t;
    }
    return n;
}

void powermethod (gsl_matrix_complex *A, gsl_vector_complex *v, unsigned int D){
    /*Compute the eigenvector whose eigenvalue is lower using the inverse power
        method*/
    int i, j, sig=1;
    double n;
    gsl_matrix_complex *L = gsl_matrix_complex_alloc(D, D);
    gsl_matrix_complex *M = gsl_matrix_complex_alloc(D, D);
    gsl_vector_complex *w = gsl_vector_complex_alloc(D);
    double d = matrixnorm(A)*eps;
    for(i=0; i<D; i++){
        for(j=0; j<D; j++){
            if(i==j){
                gsl_matrix_complex_set(L, i, i,
                    gsl_complex_add(gsl_matrix_complex_get(A, i, i),
                    gsl_complex_rect(d, 0)));
            }else{
                gsl_matrix_complex_set(L, i, j, gsl_matrix_complex_get(A, i, j));
            }
        }
    }
    gsl_permutation *p = gsl_permutation_alloc(D);
    gsl_linalg_complex_LU_decomp(L, p, &sig);
    gsl_linalg_complex_LU_invert(L, p, M);
    gsl_matrix_complex_free(L);
    gsl_permutation_free(p);

    for(i=0; i<19; i++){
        gsl_blas_zgemv(CblasNoTrans, GSL_COMPLEX_ONE, M, v, GSL_COMPLEX_ZERO, w);

        n = vectornorm(w);
        gsl_vector_complex_scale(w, gsl_complex_inverse(gsl_complex_rect(n, 0)));

        gsl_blas_zgemv(CblasNoTrans, GSL_COMPLEX_ONE, A, w, GSL_COMPLEX_ZERO, v);
        n = vectornorm(v);
        for(j=0; j<D; j++){
            gsl_vector_complex_set(v, j, gsl_vector_complex_get(w, j));
        }
        if(n< eps)break;
    }
    gsl_matrix_complex_free(M);
    gsl_vector_complex_free(w);
}

double energydensity(gsl_matrix_complex *H, gsl_matrix_complex *R,
```

```
    gsl_matrix_complex *rho, double c, unsigned int D){
    gsl_matrix_complex *Q = gsl_matrix_complex_alloc(D, D);
    gsl_matrix_complex *C = gsl_matrix_complex_alloc(D, D);
    gsl_matrix_complex *aux = gsl_matrix_complex_alloc(D, D);
    gsl_matrix_complex_memcpy(Q, H);
    gsl_blas_zgemm(CblasConjTrans, CblasNoTrans, gsl_complex_rect(-0.5, 0), R, R,
        gsl_complex_rect(0, -1), Q);
    gsl_blas_zgemm(CblasNoTrans, CblasNoTrans, GSL_COMPLEX_ONE, Q, R,
        GSL_COMPLEX_ZERO, C );
    gsl_blas_zgemm(CblasNoTrans, CblasNoTrans, gsl_complex_rect(-1, 0), R, Q,
        GSL_COMPLEX_ONE, C );
    gsl_blas_zgemm(CblasNoTrans, CblasNoTrans, GSL_COMPLEX_ONE, R, R,
        GSL_COMPLEX_ZERO, aux);
    gsl_blas_zgemm(CblasConjTrans, CblasNoTrans, GSL_COMPLEX_ONE, aux, aux,
        GSL_COMPLEX_ZERO, Q);
    gsl_matrix_complex_scale(Q, gsl_complex_rect(c, 0));
    gsl_blas_zgemm(CblasConjTrans, CblasNoTrans, GSL_COMPLEX_ONE, C, C,
        GSL_COMPLEX_ZERO, aux);
    gsl_matrix_complex_add(Q, aux);
    gsl_blas_zgemm(CblasNoTrans, CblasNoTrans, GSL_COMPLEX_ONE, rho, Q,
        GSL_COMPLEX_ZERO, aux);
    gsl_complex z = GSL_COMPLEX_ZERO;
    unsigned int i;
    for(i=0; i<D; i++){
        z = gsl_complex_add(z, gsl_matrix_complex_get(aux, i, i));
    }
    gsl_matrix_complex_free(Q);
    gsl_matrix_complex_free(C);
    gsl_matrix_complex_free(aux);
    return GSL_REAL(z);
}

double particlenumber(gsl_matrix_complex *R, gsl_matrix_complex *rho, unsigned
    int D){
    gsl_matrix_complex *aux = gsl_matrix_complex_alloc(D, D);
    gsl_matrix_complex *C = gsl_matrix_complex_alloc(D, D);
    gsl_blas_zgemm(CblasConjTrans, CblasNoTrans, GSL_COMPLEX_ONE, R, R,
        GSL_COMPLEX_ZERO, aux);
    gsl_blas_zgemm(CblasNoTrans, CblasNoTrans, GSL_COMPLEX_ONE, aux, rho,
        GSL_COMPLEX_ZERO, C);
    gsl_complex z = GSL_COMPLEX_ZERO;
    unsigned int i;
    for(i=0; i<D; i++){
        z = gsl_complex_add(z, gsl_matrix_complex_get(C, i, i));
    }
    gsl_matrix_complex_free(C);
    gsl_matrix_complex_free(aux);
    return GSL_REAL(z);
}

int gsl_linalg_complex_QR_decomp (gsl_matrix_complex * A){
    /*Compute the QR decomposition of a complex matrix
    Code by Jonny Taylor:
        http://lists.gnu.org/archive/html/help-gsl/2008-01/msg00079.html */
  const size_t M = A->size1;
```

```
  const size_t N = A->size2;
  gsl_vector_complex * tau = gsl_vector_complex_alloc(GSL_MIN (M, N));
      size_t i;

      for (i = 0; i < GSL_MIN (M, N); i++)
        {
          /* Compute the Householder transformation to reduce the j-th
             column of the matrix to a multiple of the j-th unit vector */

          gsl_vector_complex_view c_full = gsl_matrix_complex_column (A, i);
          gsl_vector_complex_view c = gsl_vector_complex_subvector
              (&(c_full.vector), i, M-i);

          gsl_complex tau_i =
              gsl_complex_conjugate(gsl_linalg_complex_householder_transform
              (&(c.vector)));

          gsl_vector_complex_set (tau, i, tau_i);

          /* Apply the transformation to the remaining columns and
             update the norms */

          if (i + 1 < N)
            {
              gsl_matrix_complex_view m = gsl_matrix_complex_submatrix (A, i, i +
                  1, M - i, N - (i + 1));
              gsl_linalg_complex_householder_hm (tau_i, &(c.vector), &(m.matrix));
            }
        }

      return GSL_SUCCESS;

}
```

B.2. Cálculo de funciones de correlación

```
#include <stdio.h>
#include <stdlib.h>
#include <gsl/gsl_math.h>
#include <gsl/gsl_rng.h>//Random number generator
#include <gsl/gsl_vector.h> //Vector handling
#include <gsl/gsl_matrix.h> //Matrix handling
#include <gsl/gsl_complex.h> //Complex numbers
#include <gsl/gsl_complex_math.h> //Complex number operations
#include <gsl/gsl_blas.h> //Linear algebra engine
#include <gsl/gsl_linalg.h>

double eps = 1e-6;

void reprM(gsl_matrix_complex *R0, unsigned int dim);
void reprv(gsl_vector_complex *v);
void extract(double *x, gsl_matrix_complex *H0, gsl_matrix_complex *R0, unsigned
    int D);
void expm(gsl_matrix_complex *A, gsl_complex alpha, int dimx);
```

```
void kronecker(gsl_matrix_complex *A, gsl_matrix_complex *B, unsigned int D,
    gsl_matrix_complex *prod);
void kronAI (gsl_matrix_complex *A, unsigned int D, gsl_matrix_complex *prod);
void kronIA (gsl_matrix_complex *A, unsigned int D, gsl_matrix_complex *prod);
void steadystatefinder(gsl_matrix_complex *H, gsl_matrix_complex *R,
    gsl_vector_complex *rho, unsigned int D);
double matrixnorm(gsl_matrix_complex *A);
double vectornorm(gsl_vector_complex *v);
void powermethod (gsl_matrix_complex *A, gsl_vector_complex *v, unsigned int D);
void spre(gsl_matrix_complex *A,  gsl_matrix_complex *preA, unsigned int D);
void spost(const gsl_matrix_complex *A, gsl_matrix_complex *postA, unsigned int
    trans, unsigned int D);

int main(){
    int D = 6, j, k;
    double x[2*D*D], dx=0.01;
    gsl_complex cff, cdd;

    FILE *f;
    f = fopen("KR_D6_c02.txt", "rt");
    for(j=0; j<2*(D)*(D); j++){
        fscanf(f, "%lf", &x[j]);
    }
    fclose(f);

    gsl_matrix_complex *H = gsl_matrix_complex_alloc(D, D);
    gsl_matrix_complex *R = gsl_matrix_complex_alloc(D, D);
    gsl_matrix_complex *Q = gsl_matrix_complex_alloc(D, D);
    gsl_vector_complex *rho = gsl_vector_complex_alloc(D*D);
    gsl_vector_complex *vaux = gsl_vector_complex_alloc(D*D);
    gsl_vector_complex *vff = gsl_vector_complex_alloc(D*D);
    gsl_vector_complex *vdd = gsl_vector_complex_alloc(D*D);
    gsl_matrix_complex *T = gsl_matrix_complex_alloc(D*D, D*D);
    gsl_matrix_complex *RR = gsl_matrix_complex_alloc(D*D, D*D);
    gsl_matrix_complex *RI = gsl_matrix_complex_alloc(D*D, D*D);
    gsl_matrix_complex *IR = gsl_matrix_complex_alloc(D*D, D*D);
    gsl_matrix_complex *A = gsl_matrix_complex_alloc(D*D, D*D);
    gsl_matrix_complex *B = gsl_matrix_complex_alloc(D*D, D*D);

    extract(x, H, R, D);
    steadystatefinder(H, R, rho, D);

    gsl_matrix_complex_memcpy(Q, H);
    gsl_blas_zgemm(CblasConjTrans, CblasNoTrans, gsl_complex_rect(-0.5, 0), R, R,
        gsl_complex_rect(0, -1), Q);
    kronecker(R, R, D, RR);
    kronAI(Q, D, T);
    kronIA(Q, D, IR);
    gsl_matrix_complex_add(T, IR);
    gsl_matrix_complex_add(T, RR);
    kronAI(R, D, RI);
    kronIA(R, D, IR);

    gsl_blas_zgemv(CblasNoTrans, GSL_COMPLEX_ONE, IR, rho, GSL_COMPLEX_ZERO, vff);
    gsl_blas_zgemv(CblasNoTrans, GSL_COMPLEX_ONE, RR, rho, GSL_COMPLEX_ZERO, vdd);
```

```
    expm(T, gsl_complex_rect(dx, 0), D*D);

    f = fopen("corr_D6_c02.txt", "wt");
    gsl_matrix_complex_memcpy(A, T);
    for(j=0; j<5000; j++){
        gsl_blas_zgemm(CblasNoTrans, CblasNoTrans, GSL_COMPLEX_ONE, T, A,
            GSL_COMPLEX_ZERO, B);

        gsl_blas_zgemv(CblasNoTrans, GSL_COMPLEX_ONE, B, vff, GSL_COMPLEX_ZERO,
            rho);
        gsl_blas_zgemv(CblasNoTrans, GSL_COMPLEX_ONE, RI, rho, GSL_COMPLEX_ZERO,
            vaux);
        cff = GSL_COMPLEX_ZERO;
        for(k=0; k<D*D; k+=D+1){
            cff = gsl_complex_add(cff, gsl_vector_complex_get(vaux, k));
        }

        gsl_blas_zgemv(CblasNoTrans, GSL_COMPLEX_ONE, B, vdd, GSL_COMPLEX_ZERO,
            rho);
        gsl_blas_zgemv(CblasNoTrans, GSL_COMPLEX_ONE, RR, rho, GSL_COMPLEX_ZERO,
            vaux);
        cdd = GSL_COMPLEX_ZERO;
        for(k=0; k<D*D; k+=D+1){
            cdd = gsl_complex_add(cdd, gsl_vector_complex_get(vaux, k));
        }

        fprintf(f, "%f\t%f\t%f\n", j*dx, GSL_REAL(cff), GSL_REAL(cdd));

        gsl_matrix_complex_memcpy(A, B);
    }
    fclose(f);
    return 0;
}

void reprM(gsl_matrix_complex *R0, unsigned int dim){
    // Auxiliar function to print matrices
    unsigned int m, n;
    for(m=0; m<dim; m++){
        for(n=0; n<dim; n++){
            printf("%f+%lfi\t", GSL_REAL(gsl_matrix_complex_get(R0, m, n)),
                GSL_IMAG(gsl_matrix_complex_get(R0, m, n)));
        }
        printf("\n");
    }
    printf("\n");
}

void reprv(gsl_vector_complex *v){
    unsigned int D = v->size, i;
    for(i=0; i<D; i++){
        printf("%.3lf+%.3lfi\t", GSL_REAL(gsl_vector_complex_get(v, i)),
            GSL_IMAG(gsl_vector_complex_get(v, i)));
    }
    printf("\n");
```

```
}

void extract(double *x, gsl_matrix_complex *H0, gsl_matrix_complex *R0, unsigned
    int D){
    unsigned int a,n,m;
    for(m=0; m<D;m++){
        gsl_matrix_complex_set(H0, m, m, gsl_complex_rect(x[m],0));
    }
    a = D;
    for(m=0; m<D;m++){
        for(n=m+1; n<D;n++){
            gsl_matrix_complex_set(H0, m, n, gsl_complex_rect(x[a], x[a+1]));
            gsl_matrix_complex_set(H0, n, m, gsl_complex_rect(x[a], -x[a+1]));
            a += 2;
        }
    }

    a=0;
    for(m=0;m<D;m++){
        for(n=0;n<D;n++){
            gsl_matrix_complex_set(R0, n, m, gsl_complex_rect(x[a+D*D],0));
            a++;
        }
    }
}

void spre(gsl_matrix_complex *A,  gsl_matrix_complex *preA, unsigned int D){
    gsl_matrix_complex_set_zero(preA);
    unsigned int i;
    for(i=0; i<D;i++){
        gsl_matrix_complex_view block =
            gsl_matrix_complex_submatrix(preA,i*D,i*D,D,D);
        gsl_matrix_complex_memcpy(&block.matrix,A);
    }
}

void spost(const gsl_matrix_complex *A, gsl_matrix_complex *postA, unsigned int
    trans, unsigned int D){
    gsl_matrix_complex_set_zero(postA);
    unsigned int i, n, m;
    gsl_complex d;
    for(m=0;m<D;m++){
        for(n=0;n<D;n++){
            if(trans){
                d = gsl_complex_conjugate(gsl_matrix_complex_get(A, n, m));
            }else{
                d = gsl_matrix_complex_get(A, m, n);
            }
            for(i=0;i<D;i++){
                gsl_matrix_complex_set(postA,i+D*n, i+D*m, d);
            }
        }
    }
}
```

```
void expm(gsl_matrix_complex *A, gsl_complex alpha, int dimx){
    /*Code by Sijo Joseph, http://stackoverflow.com/a/14718470 */
    int j,k=0;
    gsl_complex temp;
    gsl_matrix *matreal =gsl_matrix_alloc(2*dimx,2*dimx);
    gsl_matrix *expmatreal =gsl_matrix_alloc(2*dimx,2*dimx);
    //Converting the complex matrix into real one using A=[Areal,
        Aimag;-Aimag,Areal]
    for (j = 0; j < dimx;j++){
        for (k = 0; k < dimx;k++){
            temp=gsl_complex_mul(gsl_matrix_complex_get(A,j,k), alpha);
            gsl_matrix_set(matreal,j,k,GSL_REAL(temp));
            gsl_matrix_set(matreal,dimx+j,dimx+k,GSL_REAL(temp));
            gsl_matrix_set(matreal,j,dimx+k,GSL_IMAG(temp));
            gsl_matrix_set(matreal,dimx+j,k,-GSL_IMAG(temp));
        }
    }

    gsl_linalg_exponential_ss(matreal,expmatreal,.01);

    double realp;
    double imagp;
    for (j = 0; j < dimx;j++){
        for (k = 0; k < dimx;k++){
            realp=gsl_matrix_get(expmatreal,j,k);
            imagp=gsl_matrix_get(expmatreal,j,dimx+k);
            gsl_matrix_complex_set(A,j,k,gsl_complex_rect(realp,imagp));
        }
    }
    gsl_matrix_free(matreal);
    gsl_matrix_free(expmatreal);
}

void kronecker(gsl_matrix_complex *A, gsl_matrix_complex *B, unsigned int D,
    gsl_matrix_complex *prod){
    /*Compute Kronecker product A \otimes B* */
    int i, j, k, l;
    gsl_complex d;

    for (i = 0; i < D; i++){
        for (j = 0; j < D; j++){
            d = gsl_matrix_complex_get (A, i, j);
            for (k = 0; k < D; k++){
                for (l = 0; l < D; l++){
                    gsl_matrix_complex_set (prod, D*i+k, D*j+l,
                        gsl_complex_mul(d,
                        gsl_complex_conjugate(gsl_matrix_complex_get (B, k, l)))
                        );
                }
            }
        }
    }
}

void kronAI (gsl_matrix_complex *A, unsigned int D, gsl_matrix_complex *prod){
```

```
    /*Compute Kronecker product A \otimes id */
    int i, j, k;
    gsl_complex d;

    for(i=0; i<D; i++){
        for(j=0; j<D; j++){
            d = gsl_matrix_complex_get(A, i, j);
            for(k=0; k<D; k++){
                gsl_matrix_complex_set(prod, D*i+k, D*j+k, d);
            }
        }
    }
}

void kronIA (gsl_matrix_complex *A, unsigned int D, gsl_matrix_complex *prod){
    /* Compute Kronecker product id \otimes A* */
    int i, j, k;
    gsl_complex d;

    for(i=0; i<D; i++){
        for(j=0; j<D; j++){
            d = gsl_complex_conjugate(gsl_matrix_complex_get(A, i, j));
            for(k=0; k<D; k++){
                gsl_matrix_complex_set(prod, i+k*D, j+k*D, d);
            }
        }
    }
}

void steadystatefinder(gsl_matrix_complex *H, gsl_matrix_complex *R,
    gsl_vector_complex *rho, unsigned int D){
    int i=0, m, n;
    gsl_complex tr = GSL_COMPLEX_ZERO;
    gsl_rng *rng = gsl_rng_alloc(gsl_rng_taus);
    gsl_matrix_complex *L = gsl_matrix_complex_alloc(D*D, D*D);
    gsl_matrix_complex *post = gsl_matrix_complex_alloc(D*D, D*D);
    gsl_matrix_complex *pre = gsl_matrix_complex_alloc(D*D, D*D);
    gsl_matrix_complex *RtR = gsl_matrix_complex_alloc(D, D);
    spre(H,L, D);
    spost(H, post, 0, D);
    gsl_matrix_complex_sub(L,post);
    spre(R, pre, D);
    spost(R, post, 1, D);
    gsl_blas_zgemm(CblasNoTrans, CblasNoTrans, gsl_complex_rect(1,0), pre,post,
        gsl_complex_rect(0, -1), L);
    gsl_blas_zgemm(CblasConjTrans, CblasNoTrans, gsl_complex_rect(1,0), R, R,
        gsl_complex_rect(0,0), RtR);
    spre(RtR, pre, D);
    spost(RtR, post, 0, D);
    gsl_matrix_complex_add(pre, post);
    gsl_matrix_complex_scale(pre, gsl_complex_rect(0.5,0));
    gsl_matrix_complex_sub(L, pre);

    gsl_matrix_complex_free(pre);
    gsl_matrix_complex_free(RtR);
```

```
    gsl_matrix_complex_free(post);

    for(i=0; i<rho->size; i++){
        gsl_vector_complex_set(rho, i,
            gsl_complex_rect(gsl_rng_uniform(rng),gsl_rng_uniform(rng)));
    }
    powermethod(L, rho, D*D);
    gsl_matrix_complex_free(L);
    gsl_rng_free(rng);

    i=0;
    for(m=0; m<D*D; m+= (D+1)){
        tr= gsl_complex_add(tr, gsl_vector_complex_get(rho, m));
    }
    gsl_vector_complex_scale(rho, gsl_complex_inverse(tr));
}

double matrixnorm(gsl_matrix_complex *A){
    /*Compute the inf-norm of matrix A*/
    const size_t M = A->size1;
    double n = 0, t;
    int i, j;
    for(i=0; i<M; i++){
        for(j=0; j<M; j++){
            t = gsl_complex_abs(gsl_matrix_complex_get(A, i, j));
            if(t > n) n = t;
        }
    }
    return n;
}

double vectornorm(gsl_vector_complex *v){
    /*Compute the inf-norm of vector v*/
    const size_t M = v->size;
    double n = 0, t;
    int i;
    for(i=0; i<M; i++){
            t = gsl_complex_abs(gsl_vector_complex_get(v, i));
            if(t > n) n = t;
    }
    return n;
}

void powermethod (gsl_matrix_complex *A, gsl_vector_complex *v, unsigned int D){
    /*Compute the eigenvector whose eigenvalue is lower using the inverse power
        method*/
    int i, j, sig=1;
    double n;
    gsl_matrix_complex *L = gsl_matrix_complex_alloc(D, D);
    gsl_matrix_complex *M = gsl_matrix_complex_alloc(D, D);
    gsl_vector_complex *w = gsl_vector_complex_alloc(D);
    double d = matrixnorm(A)*eps;
    for(i=0; i<D; i++){
        for(j=0; j<D; j++){
            if(i==j){
```

```
                gsl_matrix_complex_set(L, i, i,
                    gsl_complex_add(gsl_matrix_complex_get(A, i, i),
                    gsl_complex_rect(d, 0)));
            }else{
                gsl_matrix_complex_set(L, i, j, gsl_matrix_complex_get(A, i, j));
            }
        }
    }
    gsl_permutation *p = gsl_permutation_alloc(D);
    gsl_linalg_complex_LU_decomp(L, p, &sig);
    gsl_linalg_complex_LU_invert(L, p, M);
    gsl_matrix_complex_free(L);
    gsl_permutation_free(p);

    for(i=0; i<19; i++){
        gsl_blas_zgemv(CblasNoTrans, GSL_COMPLEX_ONE, M, v, GSL_COMPLEX_ZERO, w);

        n = vectornorm(w);
        gsl_vector_complex_scale(w, gsl_complex_inverse(gsl_complex_rect(n, 0)));

        gsl_blas_zgemv(CblasNoTrans, GSL_COMPLEX_ONE, A, w, GSL_COMPLEX_ZERO, v);
        n = vectornorm(v);
        for(j=0; j<D; j++){
            gsl_vector_complex_set(v, j, gsl_vector_complex_get(w, j));
        }
        if(n< eps)break;
    }
    gsl_matrix_complex_free(M);
    gsl_vector_complex_free(w);
}
```