Oscilador armónico cuántico unidimensional en baño térmico usando el algoritmo Metrópolis.

Juan Esteban Aristizabal Zuluaga
Instituto de Física, Universidad de Antioquia.
(Dated: 9 de abril de 2020)

En este artículo presentamos un estudio del oscilador armónico cuántico unidimensional en un baño térmico. En particular, nos interesamos por calcular la probabilidad de encontrar el sistema en una posición dada. Para esto, presentamos los cálculos teóricos cuánticos y su contraparte clásica, con el fin de comparar los resultados. Especialmente nos enfocamos en usar el algoritmo Metrópolis para reconstruir histogramas que representan las distribuciones de probabilidad cuánticas, tanto en el espacio de posiciones como en los niveles de energía. Estos resultados cuánticos los usamos para contrastarlos con los clásicos y los tres casos presentados –baja, media y alta temperatura– concuerdan claramente con los cálculos teóricos. Se presenta también la implementación del algoritmo Metrópolis en el lenguaje de programación «Python».

Palabras clave: Oscilador armónico, física estadística cuántica, baño térmico, ensamble canónico, algoritmo Montecarlo.

I. INTRODUCCIÓN

El oscilador armónico ha sido históricamente para la física un sistema simple pero del que se puede extraer gran cantidad de información y con el que se han descubierto muchos nuevos métodos y hasta teorías completas, basadas en los razonamientos y el conocimiento obtenido de éste. Por citar un ejemplo, está la cuantización del campo electromagnético que se puede reducir a un sistema «osciladores armónicos» no acoplados y en general las teorías de segunda cuantización en la base número usan gran parte del formalismo del oscilador armónico cuántico, aunque con un significado muy diferente al que se le da en el sistema que nos compete[1, 2].

En nuestro caso hemos tomado el oscilador armónico unidimensional inmerso en un baño térmico y hemos estudiado su comportamiento a diferentes temperaturas y contrastado los resultados cuántico y clásico para la probabilidad de encontrarlo en una posición dada. Para ello hemos revisado los resultados teóricos. Entre diferentes alternativas presentadas en la literatura para llegar al resultado cuántico, entre ellas el formalizmo de intrgrales de camino [3, 4], propagadores [5] y métodos más heurísticos como el de Feynman [6], hemos decidido presentar el formalismo desarrollado por Cohen-Tannoudji [7], el cual deriva una ecuación diferencial parcial para encontrar los elementos de matriz diagonales del operador densidad, los cuales corresponden con la probabilidad en la que estamos interesados. El método que presentamos tiene la ventaja de que requiere de cálculos básicos y no de métodos avanzados, que pueden ser un poco más confusos.

Por otro lado, como sabemos, en la física estadística las herramientas computacionales han permitido un mejor entendimiento de diversos problemas. En particular, el algoritmo Metrópolis ha sido ámpliamente usado desde que Metropolis *et al.* publicaron el artículo que lo propone [8], en el año 1953, que posteriormente ganó más popularidad con la generalización hecha en 1970 por Hastings [9]. El algoritmo es útil especialmente en problemas de alta dimensionalidad para muestrear distribuciones de probabilidad en el que otros métodos no son igual de eficientes o simplemente no funcionan –uno de los ejemplos más comunes es la implementación de este algoritmo en sistemas tipo Ising [10]–, aunque en teoría se puede usar para sistemas con cualquier dimensionalidad.

En nuestro caso, a pesar de tener un sistema de baja dimensionalidad, usamos el algoritmo metrópolis para obtener los histogramas de las densidades de probabilidad para el caso cuántico tanto para $T = 0$ como varios valores de $T \neq 0$. Por otro lado, usando el mismo algoritmo, encontramos histogramas para los niveles de energía en cada caso y comprobamos que corresponden con la distribución de Boltzmann *i.e.* la distribución de probabilidad dada por el ensamble canónico de la física estadística.

La estructura del artículo es la siguiente: en la sección II presentamos los resultados teóricos para la densidad de probabilidad cuántica y clásica de encontrar el oscilador armónico unidimensional en una posición dad cuando éste está inmerso en un baño térmico. En la parte III contrastamos los resultados teóricos clásico y cuántico con simulaciones usando el algoritmo Montecarlo para la parte cuántica, para diferentes valores de temperatura. En esta sección también comprobamos los resultados y los límites de alta y baja temperatura que obtuvimos en II. En IV presentamos la conclusión del trabajo y, finalmente, en los apéndices A y B escribimos las implementaciones de los algoritmos de metrópolis usados (en Python3) para generar las figuras y para los análisis de la sección III.

II. CONSIDERACIONES TEÓRICAS

Consideraremos los sistemas en unidades reducidas, es decir, con sus variables adimensionalizadas.

A. Caso Clásico

Queremos encontrar la densidad de probabilidad de que una partícula en un potencial armónico unidimensional y en un baño térmico a temperatura $T = 1/\beta$ se encuentre en la posición x. Esta densidad de probabilidad se encuentra al integrar todas las contribuciones de los diferentes momentos de la partícula:

$$\pi^{(C)}(x;\beta) = \int_{-\infty}^{\infty} dp\rho(p,x;\beta), \tag{1}$$

donde $\rho(p,x;\beta)$ es la densidad de probabilidad clásica –en el espacio de fase– del ensamble canónico que está dada por

$$\rho(p,x;\beta) = \exp[-\beta H(p,x)]/Z(\beta), \tag{2}$$

donde

$$Z(\beta) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} dpdx \exp[-\beta H(p,x)] \tag{3}$$

es la función de partición canónica. En nuestro caso, para el oscilador armónico tenemos que el hamiltoniano está dado por

$$H(p,x) = \frac{1}{2}p^2 + \frac{1}{2}x^2. \tag{4}$$

La función de partición canónica está dada por

$$\begin{aligned} Z(\beta) &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} dpdx \exp\left[-\beta\left(\frac{p^2}{2}+\frac{x^2}{2}\right)\right] \qquad (5)\\ &= \frac{2}{\beta}\int_{-\infty}^{\infty} dye^{-y^2}\int_{-\infty}^{\infty} dze^{-z^2}\\ &= \frac{2\pi}{\beta}, \qquad (6)\end{aligned}$$

donde usamos los cambios de variable $\frac{\beta}{2}p^2 \to y^2$ $\frac{\beta}{2}x^2 \to z^2$ y el resultdo de la integral gaussiana $\int_{-\infty}^{\infty} dze^{-z^2} = \sqrt{\pi}$. De este resultado y de la ec. (2) obtenemos para el oscilador armónico

$$\rho(p,x;\beta) = \frac{\beta}{2\pi}\exp\left[-\beta\left(\frac{p^2}{2}+\frac{x^2}{2}\right)\right]. \tag{7}$$

Usando la ec. anterior en (1) obtenemos la probabilidad que buscábamos

$$\begin{aligned}\pi^{(C)}(x;\beta) =& \frac{\beta}{2\pi}\int_{-\infty}^{\infty} dp \exp\left[-\beta\left(\frac{p^2}{2}+\frac{x^2}{2}\right)\right]\\ =& \frac{\beta}{2\pi}e^{-\beta x^2/2}\int_{-\infty}^{\infty} dpe^{-\beta p^2/2}\end{aligned}$$

$$\iff \pi^{(C)}(x;\beta) = \sqrt{\frac{\beta}{2\pi}}e^{-\beta x^2/2} \tag{8}$$

Si comparamos nuestro resultado (8) con la famosa PDF gaussiana, encontramos que su varianza está dada por

$$\sigma^2 = \frac{1}{\beta} = T. \tag{9}$$

Notando esto es fácil analizar qué pasará en el caso clásico en los límites de altas y bajas temperaturas. En el primer caso, tenemos que la varianza de nuestra densidad de probabilidad sería inmensa. La probabilidad de encontrar a la partícula tiende a una distribución uniforme y si estamos trabajando en un espacio infinito $x \in (-\infty,\infty)$, en el límite $T \to \infty$ la probabilidad de encontrar a la partícula en un punto determinado sería efectivamente cero.

Por otro lado, en el límite de bajas temperaturas, la varianza de la probabilidad es tan baja como la temperatura misma. Es decir, a medida que bajamos la temperatura, el ancho de nuestra distribución gaussiana disminuye también. Esto implica que la probabilidad de encontrar la «partícula» en el centro del potencial *i.e.* en $x = 0$ aumenta conforme T disminuye, hasta que en el límite $T \to 0$ la probabilidad de encontrarla en dicho punto es uno. Es decir, la partícula permanece inmóvil en dicho sitio. En términos matemáticos, decimos que la distribución de probabilidad en este límite es una delta de Dirac centrada en el mínimo del potencial.

B. Caso cuántico

En el caso cuántico, para calcular la densidad de probabilidad de encontrar a la parícula en la posición x seguiremos a Cohen-Tannoudji [7]. Primero, recordemos que en este caso el operador densidad en el ensamble canónico está dado por

$$\hat{\rho} = \exp\left[-\beta\hat{H}\right]. \tag{10}$$

De este modo es claro que la normalización de $\hat{\rho}$ está dada por la función de partición canónica

$$Z(\beta) = \mathrm{Tr}[\hat{\rho}] = \sum_n e^{-\beta E_n} \tag{11}$$

Donde $\{E_n\}_n$ son los niveles de energía del sistema y n es un índice colectivo de números cuánticos que caracterizan el espectro de $\hat{H}$. En caso de que dicho espectro sea continuo, la suma en la expresión anterior se debería cambiar por una integral sobre las energías o los índices continuos que caracterizan las energías.

La densidad de probabilidad que se busca es

$$\pi^{(Q)}(x;\beta) = \rho(x,x;\beta)/Z(\beta) = \langle x|\hat{\rho}|x\rangle/Z(\beta). \tag{12}$$

En nuestro caso el hamiltoniano está dado por

$$\hat{H} = \frac{1}{2}\hat{p}^2 + \frac{1}{2}\hat{x}^2. \tag{13}$$

Los autoestados de energía están dados por $\{|n\rangle\}_n$, donde $n \in \{0, 1, 2, 3, ...\}$ y sus autovalores están dados por

$$\hat{H}\,|n\rangle = \left(n + \frac{1}{2}\right)|n\rangle\,. \tag{14}$$

es decir, $E_n = n + 1/2$.

Con el valor de E_n y la ecuación (11) podemos calcular ahora el valor de la función de partición

$$\begin{aligned} Z(\beta) &= e^{-\beta/2}\sum_n e^{-\beta n} \\ &= e^{-\beta/2}\sum_n \left(e^{-\beta}\right)^n \\ &= \frac{e^{-\beta/2}}{1-e^{-\beta}} \\ &= \frac{1}{2\sinh(\beta/2)}. \end{aligned} \tag{15}$$

Donde en la tercera igualdad hemos usado que $0 < e^{-\beta} < 1$ y el resultado de la serie geométrica $\sum_{n=0}^{\infty} a^n = 1/(1-a)$ si $|a| < 1$

Por otro lado, como sabemos, el hamiltoniano del oscilador armónico puede ser escrito en términos de los operadores escalera a y $a^\dagger$, que están definidos como

$$a = \frac{1}{\sqrt{2}}\left(\hat{x} + i\hat{p}\right) \tag{16}$$

$$a^\dagger = \frac{1}{\sqrt{2}}\left(\hat{x} - i\hat{p}\right), \tag{17}$$

y que tienen las siguientes propiedades

$$\left[a, a^\dagger\right] = 1 \tag{18}$$

$$a\,|n\rangle = \sqrt{n}\,|n-1\rangle \tag{19}$$

$$a^\dagger\,|n\rangle = \sqrt{n+1}\,|n+1\rangle \tag{20}$$

$$a^\dagger a\,|n\rangle = n\,|n+1\rangle\,. \tag{21}$$

Se encuentra que

$$H = a^\dagger a + \frac{1}{2}. \tag{22}$$

Con este formalismo en mente, nuestro objetivo ahora es encontrar el elemento de matriz $\langle x|\hat{\rho}|x\rangle$ para así poder calcular la probabilidad $\pi(x;\beta)$ definida en (12). Tenemos

$$\begin{aligned} \langle x|\hat{\rho}|x\rangle &= \langle x|e^{-\beta\left(a^\dagger a+1/2\right)}|x\rangle \\ &= e^{-\beta/2}F_\beta(x), \end{aligned} \tag{23}$$

donde hemos definido

$$F_\beta(x) = \langle x|e^{-\beta a^\dagger a}|x\rangle\,. \tag{24}$$

Para evaluar el elemento de matriz $F_\beta(x)$ encontraremos una ecuación diferencial que lo caracterice, usando el formalismo de las transformaciones unitarias infinitesimales. Recordemos que como $\hat{p}$ es el generador de las traslaciones

$$|x + dx\rangle = (1 - i\hat{p}dx)\,|x\rangle\,, \tag{25}$$

por tanto, tenemos que

$$\begin{aligned} F_\beta(x+dx) &= \langle x+dx|e^{-\beta a^\dagger a}|x+dx\rangle \\ &= \langle x|(1+i\hat{p}dx)\,e^{-\beta a^\dagger a}\,(1-i\hat{p}dx)|x\rangle \\ &= F_\beta(x) + idx\,\langle x|\hat{p}e^{-\beta a^\dagger a} - e^{-\beta a^\dagger a}\hat{p}|x\rangle \\ &= F_\beta(x) + idx\,\langle x|\left[\hat{p}, e^{-\beta a^\dagger a}\right]|x\rangle \end{aligned} \tag{26}$$

Donde en la tercera igualdad se ignoran términos de orden $(dx)^2$, ya que estamos considerando desplazamientos infinitesimales. Por tanto, de lo anterior obtenemos la expresión

$$\frac{F_\beta(x+dx) - F_\beta(x)}{dx} = i\,\langle x|\left[\hat{p}, e^{-\beta a^\dagger a}\right]|x\rangle\,, \tag{27}$$

y tomando $\lim_{dx\to 0}$ obtenemos la ecuación diferencial deseada

$$\frac{d}{dx}F_\beta(x) = i\,\langle x|\left[\hat{p}, e^{-\beta a^\dagger a}\right]|x\rangle\,. \tag{28}$$

Ahora nuestro problema se concentra en calcular el elemento de matriz $\langle x|\left[\hat{p}, e^{-\beta a^\dagger a}\right]|x\rangle$. Para esto calcularemos primero una expresión quivalente a la del conmutador, que facilitará el cálculo. Notamos de las ecuaciones (16) y (17) que $\hat{p} = \frac{1}{i\sqrt{2}}\left(a - a^\dagger\right)$. Por tanto, tenemos que

$$\left[\hat{p}, e^{-\beta a^\dagger a}\right] = \frac{1}{i\sqrt{2}}\left[a - a^\dagger, e^{-\beta a^\dagger a}\right]. \tag{29}$$

El conmutador en la expresión anterior involucra a $a - a^\dagger$, pero quisiéramos obtener una relación que involucre a $a + a^\dagger$ que, según las expresiones (16) y (17), es proporcional a $\hat{x}$ y actua de manera simple en los kets $|x\rangle$ de la expresión (28). Para lograr ésto examinaremos la relación entre los términos $ae^{-\beta a^\dagger a}$ y $e^{-\beta a^\dagger a}a$. Dicha relación es evidente si consideramos sus elementos de matriz en la representación $\{|n\rangle\}_n$. Tenemos

$$\begin{aligned} \langle m|ae^{-\beta a^\dagger a}|n\rangle &= \sqrt{n}e^{-\beta n}\,\langle m|n-1\rangle \\ &= \sqrt{n}e^{-\beta n}\delta_{m,\,n-1}, \end{aligned} \tag{30}$$

y

$$\begin{aligned} \langle m|e^{-\beta a^\dagger a}a|n\rangle &= \sqrt{n}e^{-\beta(n-1)}\,\langle m|n-1\rangle \\ &= \sqrt{n}e^{-\beta(n-1)}\delta_{m,\,n-1}. \end{aligned} \tag{31}$$

De (30) y (31) deducimos

$$e^{-\beta a^\dagger a}a = e^{\beta}ae^{-\beta a^\dagger a}. \tag{32}$$

Es fácil ver que la expresión anterior es equivalente a

$$\left(1 - \tanh\frac{\beta}{2}\right)e^{-\beta a^\dagger a}a = \left(1 + \tanh\frac{\beta}{2}\right)ae^{-\beta a^\dagger a}. \tag{33}$$

Similarmente, tenemos

$$\begin{aligned} \langle m|a^\dagger e^{-\beta a^\dagger a}|n\rangle &= \sqrt{n+1}e^{-\beta n}\,\langle m|n+1\rangle \\ &= \sqrt{n+1}e^{-\beta n}\delta_{m,\,n+1} \end{aligned} \tag{34}$$

y

$$\langle m|e^{-\beta a^\dagger a}a^\dagger|n\rangle = \sqrt{n+1}e^{-\beta(n+1)}\langle m|n+1\rangle = \sqrt{n+1}e^{-\beta(n+1)}\delta_{m,\,n+1} \tag{35}$$

y por tanto, tenemos que

$$e^{-\beta a^\dagger a}a^\dagger = e^{-\beta}a^\dagger e^{-\beta a^\dagger a}, \tag{36}$$

y esta expresión es equivalente a

$$\left(1+\tanh\frac{\beta}{2}\right)e^{-\beta a^\dagger a}a^\dagger = \left(1-\tanh\frac{\beta}{2}\right)a^\dagger e^{-\beta a^\dagger a}. \tag{37}$$

Restando las expresiones (33) y (37) obtenemos

$$e^{-\beta a^\dagger a}\left(a-a^\dagger\right) - \tanh\frac{\beta}{2}e^{-\beta a^\dagger a}\left(a+a^\dagger\right) = \left(a-a^\dagger\right)e^{-\beta a^\dagger a} + \tanh\frac{\beta}{2}\left(a+a^\dagger\right)e^{-\beta a^\dagger a}. \tag{38}$$

Reorganizando los términos, encontramos que

$$\left[a-a^\dagger, e^{-\beta a^\dagger a}\right] = -\tanh\frac{\beta}{2}\left[a+a^\dagger, e^{-\beta a^\dagger a}\right]_+ \tag{39}$$

donde $[A,B]_+ = AB+BA$. Además, de las relaciones (16) y (17) sabemos que $\hat{x} = \left(a+a^\dagger\right)/\sqrt{2}$ y por tanto, la expresión anterior se puede escribir como

$$\left[a-a^\dagger, e^{-\beta a^\dagger a}\right] = -\sqrt{2}\tanh\frac{\beta}{2}\left[\hat{x}, e^{-\beta a^\dagger a}\right]_+ \tag{40}$$

Remplazando la expresión anterior en (29) obtenemos

$$\left[\hat{p}, e^{-\beta a^\dagger a}\right] = i\tanh\frac{\beta}{2}\left[\hat{x}, e^{-\beta a^\dagger a}\right]_+. \tag{41}$$

Remplazando ésta última expresión en la ecuación diferencial para $F_\beta(x)$, (28), encontramos que

$$\begin{aligned} \frac{d}{dx}F_\beta(x) &= -\tanh\frac{\beta}{2}\langle x|\left[\hat{x}, e^{-\beta a^\dagger a}\right]_+|x\rangle \\ &= -\tanh\frac{\beta}{2}\langle x|\left(\hat{x}e^{-\beta a^\dagger a} + e^{-\beta a^\dagger a}\hat{x}\right)|x\rangle \\ &= -\tanh\left(\frac{\beta}{2}\right)2x\langle x|e^{-\beta a^\dagger a}|x\rangle \\ \Longleftrightarrow \frac{d}{dx}F_\beta(x) &+ \tanh\left(\frac{\beta}{2}\right)2xF_\beta(x) = 0 \end{aligned} \tag{42}$$

es fácil ver que la ecuación diferencial obtenida es de la forma $g'(x) + \frac{2x}{\zeta^2}g(x) = 0$ y que la solución de esta ecuación es una gaussiana $g(x) = c_0 e^{-x^2/\zeta^2}$ donde c_0 está definida por una condición inicial. En nuestro caso $1/\zeta^2 = \tanh(\beta/2)$. Por tanto tenemos que

$$F_\beta(x) = c_0\exp\left[-\tanh(\beta/2)x^2\right]. \tag{43}$$

Remplazando el resultado anterior en (23) encontramos que $\langle x|\hat{\rho}|x\rangle = c_0 e^{-\beta/2}\exp\left[-\tanh(\beta/2)x^2\right]$ y remplazando este resultado en (12) encontramos que la probabilidad (densidad) que buscábamos está dada por

$$\pi^{(Q)}(x;\beta) = c_0 e^{-\beta/2}\exp\left[-\tanh(\beta/2)x^2\right]/Z(\beta) \tag{44}$$

Finalmente, sabemos que al ser $\pi(x;\beta)$ una densidad de probabilidad, su integral debe ser igual a uno. Usando este hecho y el resultado de la integral gausiana dado en la sección II A, encontramos que $c_0 e^{-\beta/2}/Z(\beta) = \sqrt{\tanh(\beta/2)/\pi}$, es decir, en el ensamble canónico la probabilidad (densiadad) de encontrar a la partícula en la posición x está dada por

$$\pi^{(Q)}(x;\beta) = \sqrt{\frac{\tanh(\beta/2)}{\pi}}\exp\left[-\tanh(\beta/2)x^2\right]. \tag{45}$$

Analizar los límites de la expresión (45) también es sencillo. En el caso de bajas temperatura tenemos que $\beta \gg 1$ y por tanto $\tanh(\beta/2) \to 1$ y la densidad de probabilidad límite es $\pi^{(Q)}(x,\beta \gg 1) \approx e^{-x^2}/\sqrt{\pi}$. Este resultado corresponde con la densidad de probabilidad del estado base del oscilador armónico, $|\psi_0(x)|^2$ donde

$$\psi_0(x) = \frac{1}{\pi^{1/4}}e^{-x^2/2}, \tag{46}$$

conforme esperábamos: en el límite de bajas temperaturas los sistemas cuánticos tienden a ocupar su estado base.

Por otro lado, en el límite de altas temperaturas esperamos obtener el caso clásico. De hecho, para este límite tenemos $\beta \ll 1$ lo cual implica $\tanh(\beta/2) \to \beta/2$ y obtenemos la densidad de probabilidad clásica, $\pi^{(Q)}(x;\beta \ll 1) \approx \sqrt{\beta/2\pi}e^{-\beta x^2/2}$, que corresponde con el resultado (8).

Es interesante ver que en este ejemplo se manifiesta de manera sencilla el hecho de que en el caso de sistemas en equilibrio con reservorios de calor, los efectos de la mecánica cuántica son más evidentes en el límite de bajas temperaturas, en el cual los resultados cuántico y clásico son completamente diferentes.

Como dijimos al comienzo de esta sección, las cantidades que tratamos acá son adimensionales. Sin embargo, si queremos pasar a un sistema físico particular que se pueda reducir a un oscilador armónico en un baño térmico, en el caso cuántico podemos hacer el cambio $\beta \to \hbar\omega/k_B T$ para tener una idea de las dimensiones y casos de alta temperatura ($T \gg \hbar\omega/k_B$) o de baja temperatura ($T \ll \hbar\omega/k_B$) con valores particulares según sea el sistema considerado. En el caso clásico este paso se puede hacer haciendo el cambio $\beta \to m\omega^2 l_0^2/k_B T$, donde l_0 es una longitud característica del sistema y m es la masa de la «partícula».

III. RESULTADOS Y DISCUSIÓN

A continuación presentaremos los resultados del límite de baja temperatura para un oscilador armónico cuántico unidimensional y su contraparte clásica y de igual forma para temperaturas por encima de $T \to 0$, en el marco de simulaciones que usan cadenas de Markov en el algoritmo Metropolis. Como ya se mencionó en la Sección I,

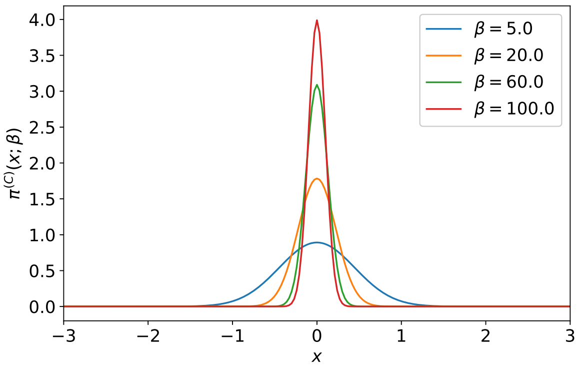

Figura 1. Probabilidad de encontrar a la partícula clásica en presencia de un potencial armónico en una posición x, cuando ésta se encuentra en un reservorio de Calor a temperatura $T = 1/\beta$. Se ilustra gráficamente el hecho de que $\pi^{(C)}(x;\beta) \to \delta(x)$ cuando $\beta \to \infty$.

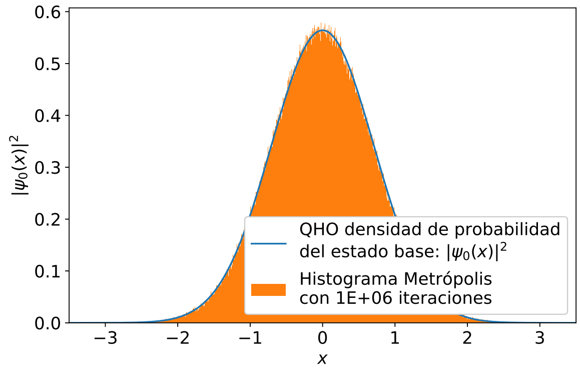

Figura 2. Densidad de probabilidad de encontrar a la «partícula» cuántica en una posición dada, cuando está en presencia de un potencial armónico y en ausencia de baño térmico *i.e.* $T = 0$. Se muestra el resultado teórico como línea continua y el histograma que resulta del algoritmo Metrópolis.

éste algoritmo permite realizar un muestreo de la densidad de probabilidad (45), hallada en la Sección II. Las implementaciones de los algoritmos en el lenguaje de programación Python3 se pueden ver en los apéndices A y B.

A. Límite de muy baja temperatura $T \to 0$

En el caso clásico y el límite de muy baja temperatura ($T \to 0$), según discutimos en la sección II A, la densidad de probabilidad de encontrar la «partícula» en la posición x tiende a una delta de Dirac: $\pi^{(C)}(x;\beta \gg 1) \to \delta(x)$. En la figura 1 encontramos $\pi^{(C)}(x;\beta)$ para diferentes valores de β, cada vez más grandes y podemos ver que para valores altos de β (temperaturas bajas) la distribución comienza a tener el límite mencionado. Aunque es solo una aproximación gráfica, podemos ver que ilustra muy bien el límite.

El límite de muy baja temperatura para el caso cuántico, como se discutió en la sección II B, es que la densidad de probabilidad $\pi^{(Q)}(x;\beta)$ tiende a la misma densidad de probabilidad del estado base del sistema, la cual determinada por la norma al cuadrado de $\psi_0(x)$, (46). En este caso usamos el algoritmo Metrópolis solo para muestrear posiciones de la distribución de probabilidad definoda por $|\psi_0(x)|$, ya que en este límite los niveles excitados del sistema no son accesibles o su accesibilidad es despreciable.

En la figura 2 mostramos la densidad de probabilidad teórica para el estado base, $|\psi_0(x)|^2$, y el histograma que obtenemos mediante el algoritmo de Metrópolis, cuya implementación se puede revisar en el apéndice A. Para obtener esta gráfica usamos 10^6 iteraciones en el algoritmo y para la propuesta de la posición en cada nueva iteración se usó una distribución uniforme centrada en el valor de x de la iteración inmediatamente anterior: $x_{new} \sim U(x_{old} - \delta x, x_{old} + \delta x)$. En este caso se usó $\delta x = 0.5$. La probabilidad de aceptación de la proposición es de

$$p(x_{old} \to x_{new}) = \text{mín}\left(1, \left|\frac{\psi_0(x_{new})}{\psi_0(x_{old})}\right|^2\right). \tag{47}$$

Podemos ver que el resultado de este ejercicio es satisfactorio. Encontramos que el histograma corresponde en gran medida con la densidad de probabilidad teórica, lo cual implica que el muestreo –a pesar de que es posible que no sea el más óptimo– si está bien realizado.

B. Temperatura finita $T \neq 0$

Para temperaturas $T \neq 0$ el oscilador armónico ahora puede acceder a niveles de energía diferentes del nivel de energía base ($n = 0$). Es por esto que la implementación del algoritmo Metrópolis en este caso debe tener en cuenta tanto cambios de posición como cambios en los niveles de energía. En la implementación, que se puede ver en el apéndice B, encontraremos que en cada iteración hay dos tipos de proposiciones: una para la posición x_{new} de manera similar que en el caso con temperatura $T = 0$, esto es, $x_{new} \sim U(x_{old} - \delta x, x_{old} + \delta x)$. La probabilidad de aceptación de esta proposición para la posición es de

$$p(x_{old} \to x_{new}) = \text{mín}\left(1, \left|\frac{\psi_{n_{old}}(x_{new})}{\psi_{n_{old}}(x_{old})}\right|^2\right). \tag{48}$$

Además, para tener en cuenta los cambios en niveles de energía dados por el contacto con el baño térmico se propone en cada nueva iteración un $n_{new} = n_{old} + \Delta n$ donde los valores $\Delta n = \pm 1$ se escogen aleatoriamente con probabilidad $1/2$. La probabilidad de aceptación de este nuevo nivel de energía está determinada por el factor de

Boltzmann y las las autofunciones de energía.

$$p(n_{old} \to n_{new}) = \\ \text{mín}\left(1, \left|\frac{\psi_{n_{new}}(x_{old})}{\psi_{n_{old}}(x_{old})}\right|^2 e^{-\beta\left(E_{n_{new}} - E_{n_{old}}\right)}\right). \tag{49}$$

En las figuras 3, 4 y 5 mostramos los resultados obtenidos mediante el algoritmo Metrópolis que describimos en el párrafo anterior, para diferentes valores de β. En esas mismas figuras se muestra también los resultados teóricos clásico y cuántico (8) y (45), respectivamente, y un histograma de las contibuciones de los diferentes niveles de energía a la densidad de probabilidad graficada, $\pi(x;\beta)$, éste último corresponde con la distribución de probabilidad

$$\pi(n;\beta) = e^{-\beta E_n}/Z(\beta), \tag{50}$$

es decir, la asociada a los niveles de energía en el ensamble canónico.

El caso de mayor temperatura en las gráficas que mostramos está en la figura 3. Éste se puede considerar como un caso de alta temperatura, lo cual podemos comprobar notando que las distribuciones de probabilidad clásica (8) y cuántica (45) para este valor de β se solapan al punto en que son casi indistinguibles en la gráfica. Esto comprueba nuestro análisis del límite de alta temperatura para $\pi^{(Q)}(x;\beta)$ que hicimos al final de la sección II B. En cuanto al histograma generado por el algoritmo se aproxima en gran medida a ambos resultados teóricos, tal como se puede observar. El histograma obtenido con el algoritmo Metrópolos en este caso para los niveles de energía se ajusta adecuadamente a la distribución de probabilidad, aunque se nota que no es del todo igual. Los niveles de energía que se muestran en el eje n son todos a los que el algoritmo accedió. Podemos notar que para altas temperaturas hay muchos niveles de energía que contribuyen significativamente a la distribución de probabilidad en el espacio de las posiciones, $\pi^{(Q)}(x;\beta)$.

El caso $\beta = 1.0$ que se ilustra en la figura 4 es un caso intermedio entre el límite de alta y baja temperatura. Aquí podemos observar perfectamente, aunque no muy grande, la diferencia entre los resultados teóricos clásico y cuántico. Encontramos también que el algoritmo Metrópolis hace bien el trabajo y se aproxima mejor a la densidad de probabilidad cuántica teórica, $\pi^{(Q)}(x;\beta)$. Aquí, el histograma para los niveles de energía generado por el algoritmo se acerca más a (50) y notamos que el número de niveles de energía que contribuyen significativamente es menor en este caso que en el caso $\beta = 0.2$ (figura 3) por un factor de aproximadamente 4.

Finalmente, en la figura 5 vemos un caso de baja temperatura, $\beta = 5$, pero evidentemente no en el cero absoluto. Aquí las diferencias entre la distribución cuántica y la clásica son muy notorias y la distribución clásica tiene una desviación estandar más pequeña que la cuántica. Recordemos que conforme la temperatura baja, el caso

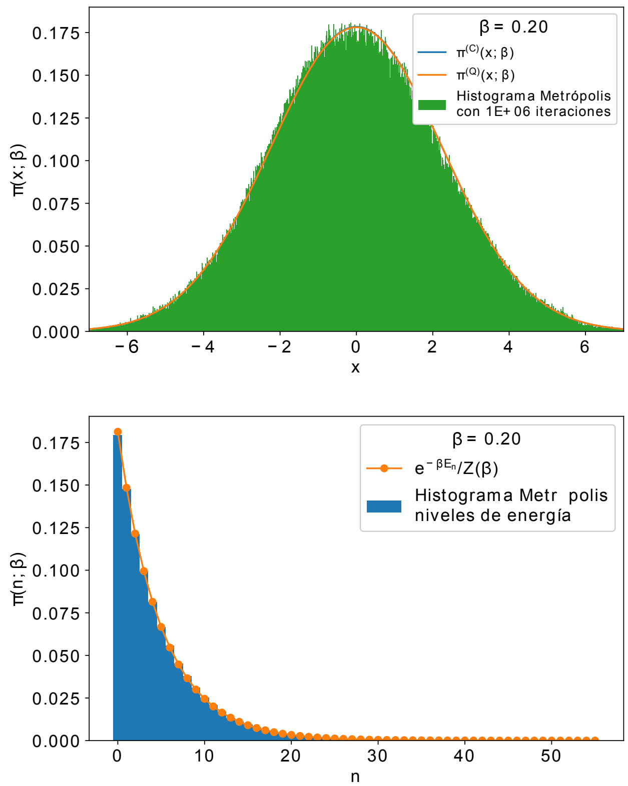

Figura 3. Arriba: densidad de probabilidad de encontrar a la «partícula» cuántica en una posición dada, cuando está en presencia de un potencial armónico y de baño térmico a temperatura definida por $\beta = 1/T = 0.2$. Mostramos los resultados teóricos clásico y cuántico como línea continua y el histograma que resulta del algoritmo Metrópolis usando 10^6 iteraciones y $\delta x = 0.5$. Observamos que éste es un límite de alta temperatura ya que las distribuciones teóricas clásica y cuántica se solapan en gran medida y son muy similares. Abajo: histograma de niveles de energía obtenido con algoritmo Metrópolis y los respectivos valores teóricos. Notamos que muchos niveles de energía contribuyen en este caso que hemos considerado de alta temperatura. Además, los valores calculados por el algoritmo se acercan en gran medida a los teóricos

clásico tiende a una desviación estandar de cero *i.e.* a una delta de Dirac. En este caso, al igual que los anteriores, el algoritmo de metrópolis muestrea correctamente la probabilidad cuántica. Aquí notamos en el histograma para n que hay una disminución importante en el número de niveles de energía que contribuyen significativamente en el sistema. Prácticamente el nivel dominante es E_0 que corresponde al estado base del sistema, que corresponde a un límite de baja temperatura, en conformidad con lo que analizamos anteriormente para el histograma de posiciones.

En cuanto al tiempo de cómputo del algoritmo, éste depende del valor de β. Conforme β disminuye (se consideran mayores temperaturas), el tiempo de cómputo aumenta ya que el sistema puede acceder a niveles de energía más altos. En el algoritmo ésto se traduce en que se deben computar los valores de ψ_n para todos los valores de x considerados hasta el momento, esto es, que hacen parte de la lista con la que se construye el histograma. En este punto tal vez el algoritmo pueda ser optimizado para reducir el tiempo de cómputo, ingeniando una manera de no tener que calcular ψ_n para todos los valores de x considerados, sino solo para los que necesiten ser usados. En nuestro caso (lo relevante más que los tiempos de cómputo son las proporciones entre dichos tiempos) el algoritmo demora aproximadamente $190s$ para el caso $\beta = 0.2$, $110s$ para $\beta = 1.0$ y $80s$ para $\beta = 5.0$. En el caso de $T = 0$ que simplificamos al muestreo de la distribución de probabilidad $|\psi_0(x)|^2$ y que tratamos en la sección III A, el algoritmo demora aproximadamente de $20s$.

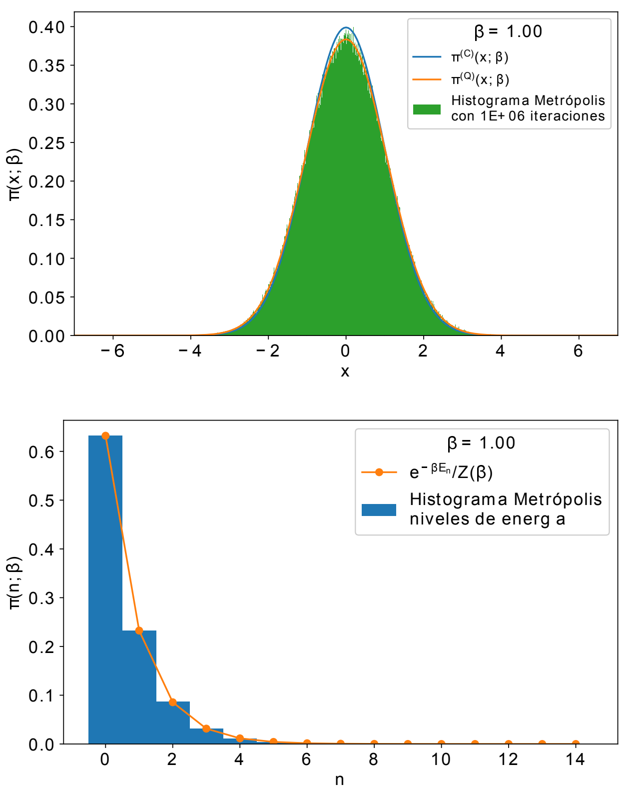

Figura 4. Arriba: densidad de probabilidad de encontrar a la «partícula» cuántica en una posición dada, cuando está en presencia de un potencial armónico y de baño térmico a temperatura definida por $\beta = 1/T = 1.0$. Mostramos los resultados teóricos clásico y cuántico como línea continua y el histograma que resulta del algoritmo Metrópolis usando 10^6 iteraciones y $\delta x = 0.5$. Observamos que éste es un caso intermedio entre los límites de alta y baja temperatura ya que las distribuciones teóricas clásica y cuántica son evidentemente diferentes aunque aún tengan cierta similitud (en los valores exactos para cada x la similitud es del orden del 90 %). Abajo: histograma de niveles de energía obtenido con algoritmo Metrópolis y los respectivos valores teóricos marcados con puntos. Este caso de temperatura media entre límite de alta y baja temperatura tiene contribuciones de muchos menos niveles de energía que el caso $\beta = 0.2$, el cual es de baja temperatura. Los valores obtenidos con el algoritmo se acercan mucho a los valoresteóricos marcados por los puntos.

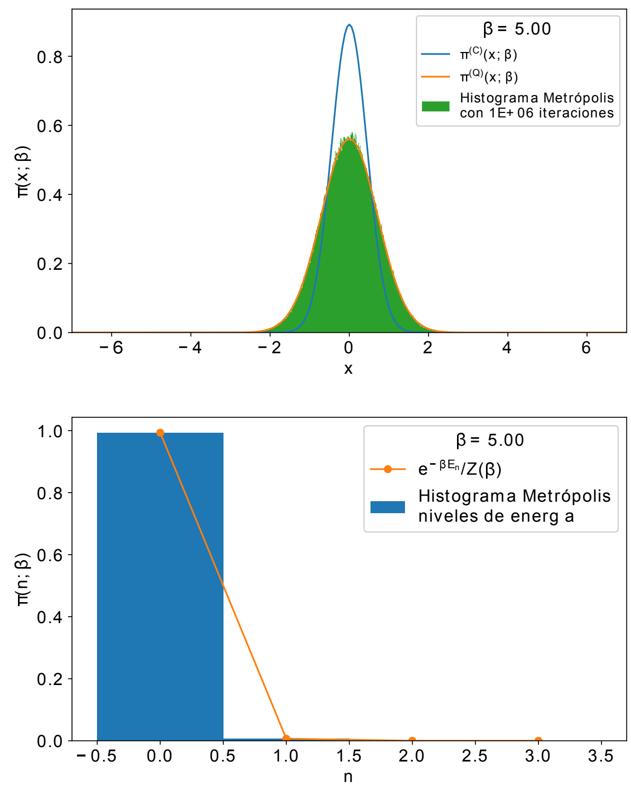

Figura 5. Arriba: densidad de probabilidad de encontrar a la «partícula» cuántica en una posición dada, cuando está en presencia de un potencial armónico y de baño térmico a temperatura definida por $\beta = 1/T = 5.0$. Mostramos los resultados teóricos clásico y cuántico como línea continua y el histograma que resulta del algoritmo Metrópolis usando 10^6 iteraciones y $\delta x = 0.5$. Observamos que éste es un límite de baja temperatura ya que las distribuciones teóricas clásica y cuántica son muy diferentes en sus valores específicos, a diferencia de lo notado en las figuras 3 y 4. Abajo: histograma de niveles de energía obtenido con algoritmo Metrópolis y los respectivos valores teóricos marcados con puntos. Aquí el histograma ya está comprendido por los niveles de energía más bajos, lo cual corresponde con el límite de baja temperatura. El nivel que más contribuye evidentemente es el estado base y los valores obtenidos con el algoritmo se ajustan casi perfectamente a los teóricos.

En cuanto a la convergencia del algoritmo a la distribución de probabilidad cuántica $\pi^{(Q)}(x;\beta)$, hay que mencionar que aunque no está reportado acá en gráficas, si se puede verificar que para valores más pequeños de β (manteniendo δx constante) el algoritmo necesita más iteraciones para converger a la densidad de probabilidad dada —en cierta medida esto se puede ver si ampliamos un poco las figuras 3, 4 y 5 ya que conforme disminuye β, las fluctuaciones en el histograma son mayores. El motivo de esto es similar al dado anteriormente: para temperaturas más altas, el sistema puede acceder a niveles de energía más altos. Esto se traduce a que se deben muestrear posiciones más alejadas de la región de mayor probabilidad ya que el acceso a niveles de energía mayores implica en últimas que la desviación estandar aumenta conforme aumenta la temperatura. En este caso se podría optimizar el algoritmo haciendo un análisis juicioso del valor de δx necesario para muestrear los posibles valores de x de manera más eficiente.

Como comentario final de esta sección, es importante también mencionar que los algoritmos se ejecutaron en Python3 v3.6.

IV. CONCLUSIÓN

En este trabajo estudiamos el problema del oscilador armónico en un baño térmico, tanto de forma clásica como cuántica y con un tratamiento teórico y computacional –éste último en el marco del algoritmo Metrópolis.

Pudimos calcular para el oscilador armónico cuántico en un baño térmico los elementos diagonales del operador densidad en la base de posiciones, $\rho(x, x; \beta)$. Estos elementos diagonales los interpretamos como la densidad de probabilidad de encontrar a la «partícula» en la posición x: $\pi^{(Q)}(x; \beta)$. En el caso clásico calculamos esta probabilidad con ayuda de la función de distribución en el espacio de fase definida por el ensamble canónico. Encontramos que el límite de baja temperatura para el caso clásico es una delta de Dirac centrada en el origen, mientras que en el caso cuántico este límite corresponde con la densidad de probabilidad de la autofunción de energía del estado base del oscilador armónico, conforme se espera. De igual modo pudimos notar que en el límite de altas temperaturas la densidad de probablididad cuántica mencionada tiende a la clásica, conforme se espera también desde la física estadística.

Para contrastar los resultados teóricos usamos el algoritmo Metrópolis para reconstruir los histogramas del sistema cuántico en el espacio de las posiciones y en los niveles de energía. Para los casos de β evaluados encontramos que uno corresponde a un límite de alta temperatura ya que las distribuciones cuántica y clásica eran muy parecidas, también tenemos un caso intermedio entre alta y baja temperatura y uno de baja temperatura. Esas conclusiones las soportamos tanto en las comparaciones de las curvas teóricas como en los histogramas generados. Siempre los histogramas de los niveles de energía corresponden con el límite que tratamos: altas temperaturas implican contribuciones apreciables de muchos niveles de energía, mientras que para bajas temperaturas contribuyen solo niveles muy próximos al estado base.

Las implementaciones de los algoritmos usados son suficientemente generales y se podrían adaptar con cierta facilidad a otros sistemas de interés que sean objeto de estudio.

AGRADECIMIENTOS

Agradezco a mis compañeros de clase con los que tuve discusiones que ayudaron en la implementación del algoritmo y en las conclusiones presentadas.

[1] G. Grynberg, A. Aspect, and C. Fabre, *Introduction to Quantum Optics: From the Semi-classical Approach to Quantized Light*, 1st ed. (Cambridge University Press, 2010).

[2] M. D. Schwartz, *Quantum Field Theory and Standard Model*, 1st ed. (Cambridge University Press, 2014) arXiv:arXiv:1011.1669v3.

[3] F. A. Barone, H. Boschi-Filho, and C. Farina, Three methods for calculating the Feynman propagator, American Journal of Physics **71**, 483 (2003), arXiv:0205085 [quant-ph].

[4] B. R. Holstein, The harmonic oscillator propagator, American Journal of Physics **66**, 583 (1998).

[5] F. Kheirandish, Exact density matrix of an oscillator-bath system: Alternative derivation, Physics Letters, Section A: General, Atomic and Solid State Physics **382**, 3339 (2018).

[6] Richard P. Feynmann, *Statistical Mechanics: a Set of Lectures*, 2nd ed. (THE BENJAMIN/CUMMINGS PUBLISHING COMPANY, INC., 1972) pp. 49–51.

[7] C. Cohen-Tannoudji, B. Diu, and F. Laloë, *Quantum mechanics* (Wiley, New York, NY, 1977) pp. 628–631.

[8] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, Equation of state calculations by fast computing machines, The Journal of Chemical Physics **21**, 1087 (1953).

[9] W. K. Hastings, Monte carlo sampling methods using Markov chains and their applications, Biometrika **57**, 97 (1970).

[10] M. E. J. Newmanand G. T. Barkema, Monte Carlo Methods in Statistical Physics, Oxford University Press , 1 (1999).

Apéndice A: Código 1: Oscilador Armónico Cuántico a muy baja temperatura $T \to 0$

```python
# -*- coding: utf-8 -*-
from __future__ import division
import numpy as np
import matplotlib.pyplot as plt
from time import time

def QHO_ground(x):
    """
    Uso: devuelve amplitud de probabilidad del estado base del Oscilador Armónico cuántico
    """
    return np.pi**(-0.25)*np.exp(-x**2/2.)

def metropolis(N=int(1e6),x0=0.0,delta=0.5,prob_amplitude_sampling=QHO_ground):
    """
    Uso: devuelve x_hist lista con N valores de x muestreados de la densidad de probabilidad
    (definida por la amplitud de probabilidad prob_amplitude_sampling) por el algoritmo
    Metrópolis.

    N: int                           ->  número de iteraciones para el algoritmo Metrópolis.
    x0: float                        ->  valor de x con el que el algoritmo inicia el muestreo.
    delta: float                     ->  tamaño máximo del paso en cada iteración de "camino
                                         aleatorio"
                                         usado por la cadena de Markov.
    prob_amplitude_sampling: func    ->  función de densidad de probabilidad a muestrear
    """
    # Iniciamos lista que almacena valores de posiciones escogidos por el algoritmo
    x_hist = [x0]
    N = int(N)
    for k in range(N):
        # Proponemos nueva posición para x con distribución uniforme centrada en valor anterior
        xnew = x_hist[-1] + np.random.uniform(-delta,delta)
        # Calculamos probabilidad de aceptancia del algoritmo Metrópolis
        acceptance_prob = min(1,(np.abs(prob_amplitude_sampling(xnew)/prob_amplitude_sampling(x_hist[-1])))**2)
        # Escogemos si aceptamos o no el valor de x propuesto
        if np.random.uniform() < acceptance_prob:
            x_hist.append(xnew)
        else:
            x_hist.append(x_hist[-1])
    return x_hist

def run_metropolis(N=1e5, x0=0.0, delta_x=0.5, prob_amplitude_sampling=QHO_ground,
                   plot=True, showplot=True, savefig=[True,'plot_QHO_ground_state.eps'],
                   xlim = 3.5, N_plot = 201):
    """
    Uso: corre el algoritmo Metrópolis que muestrea valores de x de la densidad de
    probabilidad definida por la amplitud de probabilidad prob_amplitude_sampling y
    grafica el histograma que resulta del algoritmo metrópolis, contrastado con la
    densidad de probabilidad teórica.

    Recibe:
        N: int                   -> Número de iteraciones para el algoritmo Metrópolis
        x0: float                -> valor de x con el que el algoritmo inicia el muestreo.
        delta: float             -> tamaño máximo del paso en cada iteración de "camino
                                    aleatorio"
        prob_amplitude_sampling ->  Función de densidad de probabilidad a muestrear por el
```

```
                                    algoritmo.
        showplot = True / False -> Elige si muestra o no la gráfica.
        savefig = [True / False, 'name of fig'] ->  Elige si guarda o no la gráfica.
                                                    Nombre del archivo 'name of fig'
        x_lim: float            ->  límite en x para la gráfica
        N_plot: list            ->  número de valores de x para los que se grafica densidad
                                    de probabilidad

    Devuelve:
        x_hist: list            ->  Lista con valores de x (posiciones)  obtenidos mediante
                                    cadena de Markov.
        grafica histograma y comparación con teoría si plot=True
    """
    N = int(N)
    # Corre el algoritmo metrópolis y mide tiempo de cómputo
    t_0 = time()
    x_hist = metropolis(N, x0, delta_x, prob_amplitude_sampling)
    t_1 = time()
    print('Metropolis algorithm QHO ground state: %.3f seconds for %.0E iterations'%(t_1-t_0,N))
    # Gráfica del histograma y comparación con densidad de probabilidad original
    if plot==True:
        x_plot = np.linspace(-xlim,xlim,N_plot)
        plt.figure(figsize=(8,5))
        plt.plot(x_plot,prob_amplitude_sampling(x_plot)**2,
                    label=u'QHO densidad de probabilidad\ndel estado base: $|\psi_0(x)|^2$')
        plt.hist(x_hist,bins=int(N**0.5),normed=True,
                    label=u'Histograma Metrópolis\ncon %.0E iteraciones'%(N))
        plt.xlim(-xlim,xlim)
        plt.xlabel(u'$x$')
        plt.ylabel(u'$|\psi_0(x)|^2$')
        plt.legend(loc='lower right')
        if savefig[0]==True:
            plt.savefig(savefig[1])
        if showplot==True:
            plt.show()
        plt.close()

    return x_hist

# Corremos el código usando función run_metropolis(), ésta graficará y guardará el histograma
plt.rcParams.update({'font.size':15})
x_hist = run_metropolis(N=1e6)
```

Apéndice B: Código 2: Oscilador Armónico Cuántico a temperatura finita $T \neq 0$

```
# -*- coding: utf-8 -*-
from __future__ import division
import numpy as np
import matplotlib.pyplot as plt
from time import time

def psi_0_1(x_limit = 5, N_points_x = 101):  #creates first two energy eigenfunctions
    """
    Uso:    Devuelve diccionario "psi" que representa las autofunciones de energía.
            Las llaves de "psi" están dadas por los elementos de un enmallado
            generado en el intervalo [-x_limit,x_limit] y que tiene "N_point_x" puntos
            igualmente espaciados. Los elementos asignados a cada llave x son listas
```

```
            cuyo índice corresponde al nivel de energía para la autofunción en la posición
            x.
            En pocas palabras, psi[x][n] corresponde a la autofucnión de energía \psi_{n}(x).
            Los valores accesibles para x son los elementos de grid_x y los valores
            accesibles para n son 0 y 1.

    Recibe:
        x_limit: float      ->  los valores de x serán N_points_x igualmente espaciados entre
                                [-x_limit,x_limit]
        N_ponts_x: int      ->

    Devuelve:
        psi: dict           ->  psi[x][n] corresponde a la autofucnión de energía
                                \psi_{n}(x) n = 0,1.
        grid_x: list        ->  lista con valores de x que se pueden usar en el diccionario psi.
    """
    N_points_x = int(N_points_x)
    if N_points_x%2 ==0:
        N_points_x = N_points_x + 1
    delta = x_limit/(N_points_x-1)
    grid_x = [i*delta for i in range(-int((N_points_x-1)/2),int((N_points_x-1)/2 + 1))]
    psi = {}
    for x in grid_x:
        psi[x] = [np.exp(-x**2/2.) * np.pi**(-0.25)]
        psi[x].append(2**0.5 * x * psi[x][0])
    return psi, grid_x

def add_energy_level(psi):            #adds new energy eigenfunction to psi
    """
    Uso:    Recibe diccionario generado por fucnión psi_0_1 y entrega diccionario con
            autofunciones con un nivel de energía adicional.

    Recibe:
    psi: dict       ->  diccionario con autofunciones de energía psi[x][n] y máximo
                        n = n_max = len(psi[0])

    Devuelve:
    psi: dict       -> diccionario actualizado con  máximo n = n_max + 1
    """
    # Revisamos nivel de energía máximo disponible = n-1
    n = len(psi[0.0])

    # Actualizamos diccionario de autofunciones para que contenga nivel de energía
    # inmediatamente superior al máximo accesible anteriormente (n)
    for x in psi.keys():
        psi[x].append((2./n)**0.5 * x * psi[x][n-1] -
                            ((n-1)/n)**0.5 * psi[x][n-2])
    return psi

def add_x_value(psi,x):  #adds new x value to psi
    """
    Uso:    Recibe diccionario generado por fucnión psi_0_1 y entrega diccionario con
            autofunciones con una posición adicional dada por el valor de x.

    Recibe:
    psi: dict       ->  diccionario con autofunciones de energía: psi[x][n]

```

```
    Devuelve:
    psi: dict       ->  diccionario actualizado con nueva posición accesible x para todos los
                        valores de n accesibles anteriormete.
    """
    # Añadimos primeros dos niveles de energía para la posición x (n=0 y n=1)
    psi[x] = [np.exp(-x**2/2.) * np.pi**(-0.25)]
    psi[x].append(2**0.5 * x * psi[x][0])
    #Añadimos niveles de energía superiores para la posición x:
    n_max = len(psi[0.0])-1
    for n in range(2,n_max+1):
                psi[x].append((2./n)**0.5 * x * psi[x][n-1] -
                                    ((n-1)/n)**0.5 * psi[x][n-2])
    return psi

def canonical_ensemble_prob(delta_E,beta):
    """
    Devuelve: factor de Boltzmann  para beta=1/T y delta_E dados
    """
    return np.exp(-beta * delta_E)

def boltzmann_probability(En,beta):
    """
    Recibe:
        En: float       -> autovalor de energía
        beta: float     -> inverso de temperatura en unidades reducidas beta = 1/T

    Devuelve:
        probabilidad de encontrar el oscilador armónico cuántico en nivel de energía "En"
        a tmeperatura T.
    """
    return 2.*np.sinh(beta/2.)*np.exp(-beta*En)

def metropolis_finite_temp(x0=0.0, delta_x=0.5, N=1e3,
                            prob_sampling=[psi_0_1()[0],canonical_ensemble_prob], beta=5):
    """
    Uso:    Algoritmo metrópolis para aproximar densidad de probabilidad de encontrar
            al oscilador armónico cuántico (en presencia de baño térmico) en una posición x.

    Recibe:
        x0: float       -> valor de x con el que el algoritmo inicia el muestreo.
        delta: float    -> tamaño máximo del paso en cada iteración de "camino aleatorio" .
        N: int          -> número de iteraciones para el algoritmo Metrópolis.
        prob_sampling[0]: dict      ->  diccionario con autofunciones de energía generado por
                                        la función psi_0_1().
        prob_sampling[1]: func      ->  función que calcula factor de Boltzmann.
        beta: float                 ->  inverso de temperatura en unidades reducidas beta = 1/T.


    Devuelve:
        x_hist: list    ->  lista con la que se calcula el histograma que aproxima la densidad
                            de probabilidad de encontrar al oscilador armónico cuántico (en
                            presencia de baño térmico) en una posición x.
        n_hist: list    ->  lista con la que se calcula el histograma que aproxima distribución
                            de Boltzmann para el caso del oscilador armónico cuántico.
        prob_sampling[0]: dict      ->  diccionatrio de autofunciones de energía actualizado para
                                        todos los valores de x_hist y n_hist. Se accede a ellos
                                        mediante prob_sampling[0][x][n].
```

```
    """
    # Iniciamos listas que almacenen valores de niveles de energía y posiciones escogidos
    # por el algoritmo
    x_hist = [ x0 ]
    n_hist = [ 0 ]
    prob_sampling = [prob_sampling[0].copy(),prob_sampling[1]]
    # Iniciamos iteraciones de algoritmo Metrópolis
    for k in range(int(N)):
        # Iniciamos montecarlo espacial: P(x -> x')
        x_new = x_hist[-1] + np.random.uniform(-delta_x,delta_x)
        # Revisamos si la posición propuesta x_new es accesible en el diccionario psi
        # si no es accesible, agregamos dicha posición al diccionario con respectivos
        # valores de autofunciones de energía.  Esto se hace con ayuda de la función
        # add_x_value().
        try:
            prob_sampling[0][x_new][0]
        except:
            prob_sampling[0] = add_x_value(prob_sampling[0],x_new)
        # Calculamos la probabilidad de aceptación para transiciones de posición
        # definida por algoritmo Metrópolis y se escoge si se acepta o no.
        acceptance_prob_1 = ( prob_sampling[0][x_new][n_hist[-1]] / prob_sampling[0][x_hist[-1]][n_hist[-1]] )**2
        if np.random.uniform() < min(1,acceptance_prob_1):
            x_hist.append(x_new)
        else:
            x_hist.append(x_hist[-1])

        # Iniciamos Montecarlo para nivel de energía P(n -> n')
        n_new = n_hist[-1] + np.random.choice([1,-1])
        # Chequeamos si el n propuesto es negativo
        if n_new < 0:
            n_hist.append(n_hist[-1])
        else:
            current_n_max = len(prob_sampling[0][0])-1
            # Revisamos si el nivel propuesto n_new es accesible en el diccionario psi
            # si no es accesible, agregamos dicho nivel de energía para todas las posiciones
            # del diccionario psi. Esto se hace con ayuda de la función add_energy_level().
            if n_new > current_n_max:
                prob_sampling[0] = add_energy_level(prob_sampling[0])
            # Calculamos la probabilidad de aceptación para transiciones de posición
            # definida por algoritmo Metrópolis y se escoge si se acepta o no.
            acceptance_prob_2 = ( prob_sampling[0][x_hist[-1]][n_new] / prob_sampling[0][x_hist[-1]][n_hist[-1]] )**2 * \
                                prob_sampling[1]( n_new-n_hist[-1],  beta)
            if np.random.uniform() < min(1,acceptance_prob_2):
                n_hist.append(n_new)
            else:
                n_hist.append(n_hist[-1])
    return x_hist, n_hist, prob_sampling[0]

def CHO_canonical_ensemble(x,beta=5,plot=False,savefig=True,showplot=False):
    """
    Uso:    calcula probabilidad teórica clásica de encontrar al osciladoe armónico
            (presente en un baño térmico) en la posición x. Si plot=True grafica
            dicha probabilidad.

    Recibe:
        x: float            -> posición
```

```
        beta: float         -> inverso de temperatura en unidades reducidas beta = 1/T.
        plot: bool          -> escoge si grafica o no los histogramas.
        showplot: bool      -> escoge si muestra o no la gráfica.
        savefig: bool       -> escoge si guarda o no la figura graficada.

    Devuelve:
        probabilidad teórica clásica en posición dada para temperatura T dada
        o gráfica de la probabilidad teórica clásica.
    """
    if plot==True:
        x = np.linspace(-3,3,201)
        plt.figure(figsize=(8,5))
        pdf_array = []
        for beta0 in list(beta):
            pdf_array.append( (beta0/(2.*np.pi))**0.5 * np.exp(-x**2*beta0 / 2.) )
            plt.plot(x,pdf_array[-1],label=u'$\\beta = %.1f$'%beta0)
        plt.xlim(-3,3)
        plt.xlabel('$x$')
        plt.ylabel('$\pi^{(C)}(x;\\beta)$')
        plt.legend(loc='best')
        if savefig==True:
            plt.savefig('plot_CHO_finite_temp_several_beta.eps')
        if showplot==True:
            plt.show()
        plt.close()
        return pdf_array
    else:
        return (beta/(2.*np.pi))**0.5 * np.exp(-x**2*beta / 2.)

def QHO_canonical_ensemble(x,beta):
    """
    Uso:    calcula probabilidad teórica cuántica de encontrar al osciladoe armónico
            (presente en un baño térmico) en la posición x.

    Recibe:
        x: float            -> posición
        beta: float         -> inverso de temperatura en unidades reducidas beta = 1/T.

    Devuelve:
        probabilidad teórica cuántica en posición dada para temperatura T dada.
    """
    return (np.tanh(beta/2.)/np.pi)**0.5 * np.exp(- x**2 * np.tanh(beta/2.))

def run_metropolis(psi_0_1 = psi_0_1, x_limit = 5., N_points_x = 51,
                    x0 = 0.0, delta_x = 0.5, N_metropolis = int(1e5),
                    canonical_ensemble_prob = canonical_ensemble_prob, beta = 5.,
                    plot=True, showplot = True, savefig = True, legend_loc = 'best', x_plot_0=7):
    """
    Uso:    Corre algoritmo Metrópolis para el oscilador armónico cuántico en un baño térmico.
            Grafica el histograma de posiciones obtenido contrastándolo con los resultados
            teóricos cuántico y clásico. Grafica histograma de niveles de energía visitados por
            el algoritmo.

    Recibe:
        psi_0_1: función    ->  función que inicializa las autofunciones del hamiltoniano.
        x_limit: float      ->  las autofunciones se inicializan en intervalo
↪   (-x_limit,x_limit).
        N_points_x: int     ->  la rejilla para inicializar autofunciones tiene
```

```
                                N_points_x puntos.
        x0: float           ->  valor de x con el que el algoritmo inicia el muestreo.
        delta_x: float      ->  tamaño máximo del paso en cada iteración de "camino aleatorio".
        N_metropolis: int   ->  número de iteraciones para algoritmo metrópolis.
        beta: float         ->  inverso de temperatura del baño térmico en unidades reducidas
                                beta = 1/T.
        canonical_ensemble_prob: función   ->   función que genera factor de Boltzmann
                                                exp(-B*deltaE).
        plot: bool                      ->  escoge si grafica o no los histogramas
        showplot: bool                  ->  escoge si muestra o no la gráfica
        savefig: [bool,'name of fig']   ->  escoge si guarda o no la figura y el nombre del
                                            archivo.
        legend_loc: 'position'          ->  posición de la legenda para la figura
        x_plot_0: float                 ->  dominio de la gráfica en x será (-x_plot,x_plot)

    Devuelve:
        x_hist: list        ->  Lista con valores de x (posiciones)  obtenidos mediante cadena
                                de Markov.
        n_hist: list        ->  Lista con valores de n (niveles de energía) obtenidos mediante
                                cadena de Markov.
        psi_final: dict     ->  Diccionario con autofunciones de energía \psi_{n}(x) = psi[x][n]
                                    para valores de x y n en x_hist y n_hist.
    """
    # Inicializamos autofunciones de energía en diccionario psi generado por función psi_0_1()
    psi, grid_x = psi_0_1(x_limit,N_points_x)

    # Almacenamos probs. en una lista:  la amplitud de probabilidad psi de las autofunciones
    #                                   y el factor de Boltzmann del ensamble canónico
    prob_sampling = [psi, canonical_ensemble_prob]

    # Ejecutamos algoritmo metropolis y medimos tiempo de cómputo
    t_0 = time()
    x_hist, n_hist, psi_final = metropolis_finite_temp(x0=x0, delta_x=delta_x,N=N_metropolis,
                                                    prob_sampling=prob_sampling, beta=beta)
    t_1 = time()
    print('Metropolis algorithm (beta = %.2f): %.3f seconds for %.0E iterations'%(beta,t_1-t_0,N_metropolis))

    if plot==True:
        # Graficamos histograma para posiciones
        x_plot = np.linspace(-x_plot_0,x_plot_0,251)
        plt.figure(figsize=(8,5))
        plt.plot(x_plot,CHO_canonical_ensemble(x_plot,beta=beta),
                    label=u'$\pi^{(C)}(x;\\beta)$')
        plt.plot(x_plot,QHO_canonical_ensemble(x_plot,beta=beta),
                    label=u'$\pi^{(Q)}(x;\\beta)$')
        plt.hist(x_hist,bins=int(N_metropolis**0.5),normed=True,
                    label='Histograma Metrópolis\ncon %.0E iteraciones'%(N_metropolis))
        plt.xlim(-x_plot_0,x_plot_0)
        plt.xlabel(u'$x$')
        plt.ylabel(u'$\pi(x;\\beta)$')
        plt.legend(loc=legend_loc, title=u'$\\beta=%.2f$'%beta, fontsize=12)
        plt.tight_layout()
        if savefig==True:
            plt.savefig('plot_QHO_finite_temp_beta_%d_%d.eps'%(beta,(beta-int(beta))*100))
        if showplot==True:
            plt.show()
```

```
        plt.close()

        # Graficamos histograma para niveles de energía
        n_plot = np.arange(len(psi_final[0]))
        plt.figure(figsize=(8,5))
        plt.hist(n_hist,normed=True,bins=np.arange(len(psi_final[0])+1)-0.5,
                    label='Histograma Metrópolis\nniveles de energía')
        plt.plot(n_plot,boltzmann_probability(n_plot+1/2,beta),'o-',
                    label=u'$e^{-\\beta E_n}/Z(\\beta)$')
        plt.xlabel(u'$n$')
        plt.ylabel(u'$\pi(n;\\beta)$')
        plt.legend(loc='best', title=u'$\\beta=%.2f$'%beta)
        plt.tight_layout()
        if savefig==True:
            plt.savefig('plot_QHO_n_hist_beta_%d_%d.eps'%(beta,(beta-int(beta))*100))
        if showplot==True:
            plt.show()
        plt.close()

    return x_hist, n_hist, psi_final

plt.rcParams.update({'font.size':15})

# Corremos algoritmo metrópolis usando función run_metropolis() para varios
# valores de beta
beta_array = [0.2, 1, 5, 60]
legend_loc =['lower center', 'lower right', 'best', 'best']
for i,beta in enumerate(beta_array):
    run_metropolis(N_metropolis=1e6,beta=beta,showplot=False)

# Corremos algoritmo para gráfica de límite de baja temperatura en el caso
# clásico (figura 1 en el artículo)
beta_array_CHO = [5,20,60,100]
CHO_canonical_ensemble(0,beta=beta_array_CHO,plot=True,showplot=False)
```