Partícula bajo influencia de potencial armónico y anarmónico en colectividad canónica: algoritmos *Matrix Squaring* y *Path Integral*.

Juan Esteban Aristizabal Zuluaga
Instituto de Física, Universidad de Antioquia.
(Dated: 30 de abril de 2020)

En este artículo presentamos un estudio del oscilador armónico cuántico unidimensional y de un sistema unidimencional de una partícula bajo la acción de un potencial anarmónico, ambos inmersos en un baño térmico. En particular, nos interesamos por la implementación de los algoritomos *Matrix Squaring* y *Path Integral Naive Sampling* que nos permiten obtener informacón relevante de los sistemas en cuestión. Para esto presentamos una breve descripción de los algoritmos así como los resultados obtenidos. En especial, nos enfocamos en calcular la energía promedio de los sistemas y la probabilidad de encontrar a la partícula en una posición dada. Al contrastar los resultados obtenidos con los valores teóricos –cuando es posible– encontramos que ambos algoritmos se aproximan a los valores deseados. Se presenta también la implementación de los algoritmos en el lenguaje de programación «Python».

Palabras clave: Oscilador armónico, potencial anarmónico, física estadística cuántica, baño térmico, ensamble canónico, *matrix squaring*, *path integral*.

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]. Además, los casos anarmónicos han sido igualmente importantes, ya que muchos sistemas reales se pueden modelar usando este tipo de potenciales, además que éstos se puede estudiar generalmente como una perturbación del potencial armónico.

En este trabajo estudiamos dos métodos computacionales que permiten obtener información acerca tanto del oscilador armónico unidimensional como de un sistema de una partícula bajo la acción de un potencial anarmónico unidimensional, ambos inmersos en un baño térmico.

El primer método que presentamos se conoce como *Matrix Squaring* y se basa en la convolución de matrices densidad en el ensamble canónico. Cuando este procedimiento se repite iterativamente, es posible obtener matrices densidad a bajas temperaturas a partir de matrices densidad a altas temperaturas. Ésto último es una ventaja grande ya que se puede usar la aproximación de Trotter para matrices densidad a altas temperaturas, en la cual solo es necesario conocer el potencial de interacción del sistema que se desea examinar y la matriz densidad de la partícula libre [3]. De este modo se logra obtener, sin conocer en un principio la matriz densidad del sistema en cuestión a bajas temperaturas. Ya con la matriz densidad del sistema a bajas temperaturas se pueden realizar muchos cálculos interesantes. En nuestro caso nos interesamos en calcular la probabilidad de encontrar la partícula en una posición dada, $\pi^{(Q)}(x;\beta)$, y también en encontrar la energía promedio del sistema, $\langle E\rangle$. Ambos resultados computacionales se contrastaron con los respectivos resultados teóricos y éstos se ajustan muy bien. En esta parte del trabajo (para el potencial armónico) se hizo también un análisis de los parámetros óptimos necesarios para correr el algoritmo y minimizar el error computacional, lo cual se vio reflejado satisfactoriamente en los gráficos de $\pi^{(Q)}(x;\beta)$ y $\langle E\rangle$.

Por otro lado se estudió el algoritmo *Path Integral Naive Sampling* el cual, a groso modo, consiste en obtener un histograma para las posiciones de la partícula a partir de caminos que se construyen perturbando un camino inicial propuesto y cuyas modificaciones se aceptan o niegan de acuerdo a una dinámica de metrópolis, donde cada configuración se asume con probabilidad proporcional al término del argumento de la integral de camino [3]. Esta idea se ampliará con más detalle en su momento. Los resultados de este algoritmo, aunque no tan precisos como los del algoritmo de *Matrix Squaring*, se ajustan en cierta medida a los resultados teóricos y mejoran conforme se usan más iteraciones para calcular más perturbaciones en el camino inicial propuesto.

La estructura del artículo es la siguiente: en la sección II presentamos los pormenores de los algoritmos que se usaron en el trabajo. Aquí se explica en detalle en qué consiste cada algoritmo y cómo se obtienen los resultados. En la parte III mostramos los resultados obtenidos al correr los algoritmos para el oscilador armónico y para el potencial anarmónico. En esta sección veremos las diferencias entre los resultados obtenidos mediante los diferentes algoritmos y encontraremos también que en general, coinciden con los valors teóricos conocidos. En IV presentamos la conclusión del trabajo y, finalmente, en los apéndices A y B escribimos las implementaciones de los algoritmos usados (en Python3) para generar las

figuras y para los análisis de la sección III.

II. CONSIDERACIONES TEÓRICAS

Los algoritomos *Matrix Squaring* y *Path Integral Naive Sampling* están basados en un hecho muy sencillo. Nos aprovechamos de que la matriz densidad en el ensamble canónico se escribe como

$$\hat{\rho}(\beta) = \mathrm{e}^{-\beta \hat{H}}. \tag{1}$$

De este modo es muy sencillo ver que

$$\begin{aligned}\hat{\rho}(\beta) &= \mathrm{e}^{-\beta \hat{H}/N} \cdot \mathrm{e}^{-\beta \hat{H}/N} \cdots \mathrm{e}^{-\beta \hat{H}/N} \rightarrow N \text{ veces en total} \\ &= \hat{\rho}(\beta/N) \cdot \hat{\rho}(\beta/N) \cdots \hat{\rho}(\beta/N)\end{aligned} \tag{2}$$

El significado físico de la ecuación anterior es que podemos construir matrices densidad a temperatura $1/\beta$ usando matrices densidad a temperaturas más altas N/β.

Por otro lado, en estos algoritmos también usamos la conocida aproximación de Trotter para la matriz densidad a altas temperaturas [3]

$$\rho\left(x, x', \beta\right) \underset{\beta \to 0}{\longrightarrow} \mathrm{e}^{-\frac{1}{2}\beta V(x)} \rho^{\text{free}}\left(x, x', \beta\right) \mathrm{e}^{-\frac{1}{2}\beta V\left(x'\right)}. \tag{3}$$

Con esto en mente, veamos cómo se construyen los algoritmos mencionados.

A. *Matrix Squaring*

Para enternder la idea de este aloritmo, primero consideramos la ecuación (2) con $N = 2$ y con el cambio $\beta \to 2\beta$ para construir, a partir de dos matrices densidad a temperatura $1/\beta = T$, una matriz densidad a temperatura $1/2\beta = T/2$:

$$\hat{\rho}(2\beta) = \hat{\rho}(\beta) \cdot \hat{\rho}(\beta). \tag{4}$$

En el algoritmo lo que hacemos iterar (4) N_{iter} veces hasta conseguir un valor de beta deseado. De este modo, en primer lugar usamos $\hat{\rho}(\beta_{\text{ini}})$ para obtener $\hat{\rho}(2\beta_{\text{ini}})$. Esto lo representamos como $\hat{\rho}(\beta_{\text{ini}}) \rightarrow \hat{\rho}(2\beta_{\text{ini}})$. Luego tomamos $\hat{\rho}(2\beta_{\text{ini}})$ y obtenemos $\hat{\rho}(2^2\beta_{\text{ini}})$. Al seguir iterando hasta N_{iter} veces tenemos

$$\begin{aligned}\hat{\rho}(\beta_{\text{ini}}) &\rightarrow \hat{\rho}(2\beta_{\text{ini}}) \\ \hat{\rho}(2\beta_{\text{ini}}) &\rightarrow \hat{\rho}(2^2\beta_{\text{ini}}) \\ &\cdots \\ \hat{\rho}(2^{N_{\text{iter}}-1}\beta_{\text{ini}}) &\rightarrow \hat{\rho}(2^{N_{\text{iter}}}\beta_{\text{ini}}) = \hat{\rho}(\beta_{\text{fin}})\end{aligned} \tag{5}$$

Aquí vemos el significado del nombre *Matrix Squaring*, que quier decir "elevar al cuadrado la matriz", que en efecto es lo que estamos haciendo. En retrospectiva, lo que hacemos con este algpritmo es obtener, a partr de la matriz densida $\hat{\rho}(\beta_{\text{ini}})$, la matriz $\hat{\rho}(\beta_{\text{fin}})$ donde $\beta_{\text{fin}} = 2^{N_{\text{iter}}}\beta_{\text{ini}}$. Es importante notar que si queremos obtener una matriz densidad a temperatura inversa β_{fin} y usamos $\beta_{\text{ini}} = 2^{-N_{\text{iter}}}\beta_{\text{fin}}$ con N_{iter} lo suficientemente grande, estaremos obteniendo una matriz densidad a temperatura muy baja en comparación con la usada como *input* del algoritmo. Este hecho lo podemos usar a nuestro favor, ya que en este caso podremos usar la aproximación de Trotter para altas temperaturas, como veremos a continuación.

En nuestro caso trabajamos en el espacio de las posiciones para poder llevar a cabo el algoritmo. Así, tenemos que para la primera iteración del algoritmo y usando la idea que mencionamos anteriormente, $\beta_{\text{ini}} = 2^{-N_{\text{iter}}}\beta_{\text{fin}}$, tenemos

$$\begin{aligned}&\rho(x, x'; 2^{-N_{\text{iter}}+1}\beta_{\text{fin}}) = \\ &\qquad \int dx_1 \rho(x, x_1; 2^{-N_{\text{iter}}}\beta_{\text{fin}}) \rho(x_1, x'; 2^{-N_{\text{iter}}}\beta_{\text{fin}}).\end{aligned} \tag{6}$$

si asumimos que N_{iter} es lo suficientemente grande como para considerar que las matrices de densidad de la derecha están a temperaturas altas, podemos usar la aproximación de Trotter a nuestro beneficio, obteniendo

$$\begin{aligned}&\rho(x, x'; 2^{-N_{\text{iter}}+1}\beta_{\text{fin}}) = \\ &\int dx_1 \mathrm{e}^{-\frac{1}{2}2^{-N_{\text{iter}}}\beta V(x)} \rho^{\text{free}}\left(x, x_1, 2^{-N_{\text{iter}}}\beta\right) \\ &\times \mathrm{e}^{-2^{-N_{\text{iter}}}\beta V(x_1)} \rho^{\text{free}}\left(x_1, x', 2^{-N_{\text{iter}}}\beta\right) \mathrm{e}^{-\frac{1}{2}2^{-N_{\text{iter}}}\beta V\left(x'\right)},\end{aligned} \tag{7}$$

de modo que si podemos usar esta aproximación, no necesitamos conocer la matriz densidad del sistema para inicializar el algoritmo. Basta con conocer la matriz densidad $\hat{\rho}^{\text{free}}$ y el potencial del sistema. Así, iternado la ecuación anterior un total de N_{iter} veces obtenemos, como se indica en (5), la matriz densidad $\hat{\rho}(\beta_{\text{fin}})$.

B. *Path Integral Naive Sampling*

La integral de camino en mecánica estadística se construye también con ayuda de la expresión (2). En este caso se toma esta expresión en el espacio de las posiciones

$$\begin{aligned}&\rho(x, x'; \beta) = \\ &\int dx_1 dx_2 \cdots dx_{N-1} \rho(x, x_1; \beta/N) \rho(x_1, x_2; \beta/N). \\ &\qquad \cdots \rho(x_{N-1}, x'; \beta/N)\end{aligned} \tag{8}$$

La interpretación que se le da a (8) es que la partícula viaja de x a x' a través de una serie de pasos intermedios $x_1,\ x_2,\ , \ldots, x_{N-1}$ que constituyen un camino o *path*. La amplitud total $\rho(x, x'; \beta)$ está constituída por la suma de todos los posibles caminos. Esto es, por todos los posibles valires de las posiciones intermedias x_i [4].

Cuando $N \to \infty$ se obtiene la integral de camino que usualmente se denota simbólicamente como

$$\rho\left(x, x'; \beta\right) = \int \mathscr{D}x(\beta) \Phi[x(\beta)], \tag{9}$$

donde

$$\Phi[x(\beta)] = \lim_{\substack{\varepsilon\to 0\\ N\varepsilon=\beta}} \rho\left(x, x_1; \varepsilon\right)\rho\left(x_1, x_2; \varepsilon\right)\cdots\rho\left(x_{N-1}, x'; \varepsilon\right), \tag{10}$$

y

$$\mathscr{D}x(\beta) = \lim_{N\to\infty} dx_1 dx_2 \cdots dx_{N-1}. \tag{11}$$

Sin embargo, en el caso computacional, la expresión útil es (8) ya que es necesario tener una expresión discreta para implementarla en un algoritmo.

En el algoritmo estamos interesados en calcular la probabilidad de encontrar a la partícula en una posición x, es decir, estamos interesados en los elementos de la diagonal de la matriz densidad

$$\pi^{(Q)}(x;\beta) = \rho\left(x, x; \beta\right). \tag{12}$$

Para encontrar esto usamos el siguiente algoritmo, que está basado en la idea de la integral de camino. En primer lugar proponemos un camino de tamaño N inicializado en cero

$$X = [x_1, x_2, \ldots, x_{N-1} = 0] \tag{13}$$

(se deben tomar condiciones de frontera periódicas ya que $x = x'$) de la interpretación dada por Feynmann [4], este camino tiene asociada una amplitud de probabilidad igual al argumento de la integral de camino, que en el caso antes de pasar al continuo (i.e. en la expresión (8) es

$$p(X) = \rho(x, x_1; \beta/N)\rho(x_1, x_2; \beta/N)\cdots\rho(x_{N-1}, x'; \beta/N) \tag{14}$$

Posteriormente escogemos un x_i al azar y proponemos un nuevo valor

$$x_{i\_\text{new}} = x_i + \delta x \tag{15}$$

donde

$$\delta x \sim U(-\delta, \delta), \tag{16}$$

es decir, es un número aleatoro distribuído uniformemente entre $-\delta$ y δ. El nuevo camino propuesto

$$X_{\text{new}} = [x_1,\ x_2,\ \ldots,\ x_{i\_\text{new}},\ \ldots,\ x_{N-1}] \tag{17}$$

tiene asociada una amplitud de probabilidad

$$\begin{aligned} &p(X_{\text{new}}) = \\ &\quad \rho(x, x_1; \beta/N)\rho(x_1, x_2; \beta/N)\cdots\rho(x_{i-1}, x_{i\_\text{new}}; \beta/N) \\ &\quad \rho(x_{i\_\text{new}}, x_{i+1}; \beta/N)\cdots\rho(x_{N-1}, x'; \beta/N). \end{aligned} \tag{18}$$

De este modo, podemos usar la probabilidad de aceptación del algoritmo Metrópolis para decidir si aceptamos o no el nuevo camino. Dicha probabilidad está dada por

$$\begin{aligned} &p(X \to X_{\text{new}}) = \\ &\quad \text{mín}\left(1, \frac{\rho(x_{i-1}, x_{i\_\text{new}}; \beta/N)\rho(x_{i\_\text{new}}, x_{i+1}; \beta/N)}{\rho(x_{i-1}, x_i; \beta/N)\rho(x_i, x_{i+1}; \beta/N)}\right) \end{aligned} \tag{19}$$

Este proceso se realiza de forma iterativa N_{iter} veces. En cada iteración se debe guardar el camino aceptado (o, si el valor se rechaza, guardar de nuevo el camino de la iteración inmediatamente anterior). De modo que al final del algoritmo debemos tener N_{iter} valores de cada x_i. Posteriormente, construyendo el histograma de los N_{iter} valores de posiciones x_i o de los valores de posiciones de todos los caminos unidos $x_0 \cup x_1 \cup \cdots \cup x_{N-1}$ obtendremos una aproximación a la probabilidad (12) deseada.

Hay que mencionar que en la expresión (19) se puede remplazar ρ por su correspondiente aproximación de Trotter (3) siempre que β/N corresponda con una temperatura lo suficientemente alta. Esto es lo que en efecto hacemos en nuestra implementación computacional.

III. RESULTADOS Y DISCUSIÓN

A. *Matrix Squaring*

Las integrales que aparecen en este algoritmo, explicado en la sección II A, se deben aproximar a sumatorias, y por tanto los valoresde x se deben discretizar. Computacionalmente se construye una lista $x = [-x_{\max}, -x_{\max}+dx, ..., x_{\max}-dx, x_{\max}]$ donde $x_{\max} > 0$. Los valores de dx y $\beta_{\text{ini}} = 2^{-N_{\text{iter}}}\beta_{\text{fin}}$ deben ser cuidadosamente escogidos de manera que se minimice el error computacional. Como ya se ha dicho, la cantidad en la que estaremos interesados es en $\pi^{(Q)}(x;\beta)$, que se define en (12). Afortunadamente, el valor teórico de esta cantidad para el oscilador armónico es conocido y tiene un valor de [5]

$$\pi^{(Q)}_{\text{teo}}(x;\beta) = \sqrt{\frac{\tanh(\beta/2)}{\pi}}\exp\left[-\tanh(\beta/2)x^2\right]. \tag{20}$$

Teniendo en cuenta este valor y el valor computacional $\pi^{(Q)}_{\text{comp}}(x;\beta)$ podemos construir un error relativo entre estas dos funciones

$$\delta(\beta_{\text{fin}}) = \frac{\int dx\left|\pi^{(Q)}_{\text{teo}}(x;\beta_{\text{fin}}) - \pi^{(Q)}_{\text{comp}}(x;\beta_{\text{fin}})\right|}{\int dx \pi^{(Q)}_{\text{teo}}(x;\beta_{\text{fin}})}. \tag{21}$$

De este modo cuantificamos el error computacional, que depende de los parámetros usados en cada caso.

En la figura 1 mostramos los valores de el error relativo δ para $\beta_{\text{fin}} = 4$ al variar los parámetros dx y β_{ini}. Encontramos acá un comportamiento en cierta medida esperado: si se escogen valores de temperatura inversa inicial β_{ini} lo suficientemente pequeños en conjunto con espaciamientos dx lo suficientemente pequeños, el error computacional es pequeño. Esto tiene sentido, ya que en estos límites la discretización es más suave (se parece más a un continuo) y la aproximación de Trotter se hace más 'válida' ya que la temperatura inicial es mayor. El valor que se paga por tomar estos valores pequeños es tiempo computacional, que en este caso no se reporta cuantitativamente, pero es significativamente mayor cuanto más

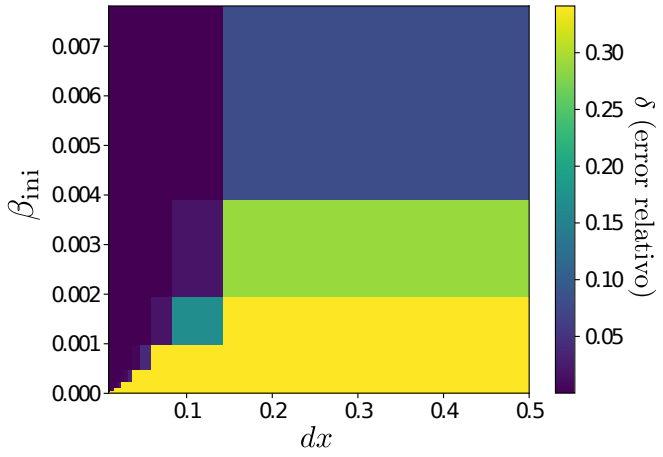

Figura 1. Error relativo cuantificado por (21) para $\beta_{\text{fin}} = 4$. Notamos que cuanto más pequeña sea β_{ini}, la discretización dx debe disminuir también para evitar que el error computacional sea grande.

pequeña es la discretización dx (asumiendo que $x_{\max}$ se deja constante).

Notamos también que cuanto más pequeño queramos que sea β_{ini}, necesitamos un valor de discretización aún más pequeño para que el error computacional no sea grande. Esto se debe a que en el momento de construir la figura nos encontramos con que hay unos valores de los parámetros dx y β_{ini} para los cuales el algoritmo no converge a una respuesta, es decir, los valores de $\pi^{(Q)}_{\text{comp}}(x;\beta_{\text{fin}})$ daban nan ó inf, lo cual implica que el error en estos casos sería 'infinito'. Sin embargo, en la gráfica lo que se asignó a estos valores fue el error máximo que se encontró en esta región, excluyendo los valores para los cuales el algoritmo no convergió a una respuesta (en este caso color amarillo). Se hizo de esta manera para que la barra de colores pudiera dar cuenta de forma acertada de todos los errores en la región de dx y β_{ini} escogida. De lo contrario, lo que se visualizaría en la gráfica sería solo color amarillo totalmente en las regiones que no convergía el algoritmo y azul totalmente en las regiones en que si convergía y los valores intermedios no se hubiesen podido visualizar. En términos prácticos, la región que está pintada en amarillo en el gráfico indica que el algoritmo no converge para los parámetros dados allí y esto se debe a que la precisión computacional no es suficiente para resolver el problema.

Para obtener $\pi^{(Q)}_{\text{comp}}(x;\beta_{\text{fin}})$ con $\beta_{\text{fin}} = 4$ nos basamos en la figura 1 para escogier los parámetros $dx = 0.05$ y $\beta_{\text{ini}} = 2^{-9}\beta_{\text{fin}} \approx 0.0078$, que como vemos en la figura mencionada, tiene un error apreciablemente pequeño, además no toma tanto tiempo computacional. Con los parámetros mencionados se construyó la figura 2 que en principio muestra tanto el valor teórico como el computacional de $\pi^{(Q)}(x;\beta)$. Las diferencias entre $\pi^{(Q)}_{\text{teo}}(x;\beta_{\text{fin}})$ y $\pi^{(Q)}_{\text{comp}}(x;\beta_{\text{fin}})$ son tan pequeñas, que en la gráfica están

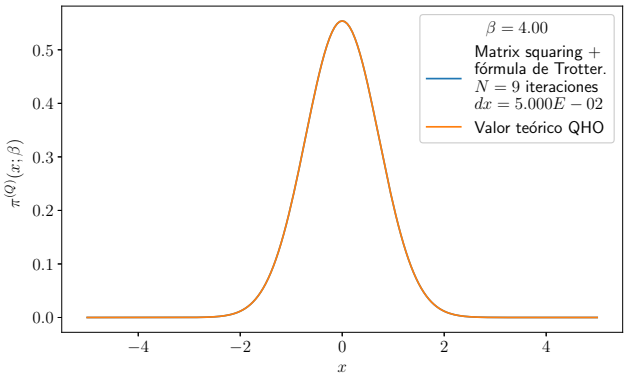

Figura 2. $\pi^{(Q)}_{\text{teo}}(x;\beta_{\text{fin}})$ y $\pi^{(Q)}_{\text{comp}}(x;\beta_{\text{fin}})$ para el oscilador armónico cuántico. En la gráfica solo aparece uno de los valores ya que el error computacional en este caso fue muy pequeño y las gráficas se solaparon. Los parámetros del algoritmo fueron $\beta_{\text{fin}} = 4$, $\beta_{\text{ini}} = 2^{-9}\beta_{\text{fin}}$, $x_{\max} = 5$, $dx = 0.05$.

solapadas completamente y solo se alcanza a ver una de ellas. Con este resultado notamos que los parámetros escogidos están bien optimizados y nos sirve para calibrar estos parámetros para el oscilador anarmónico.

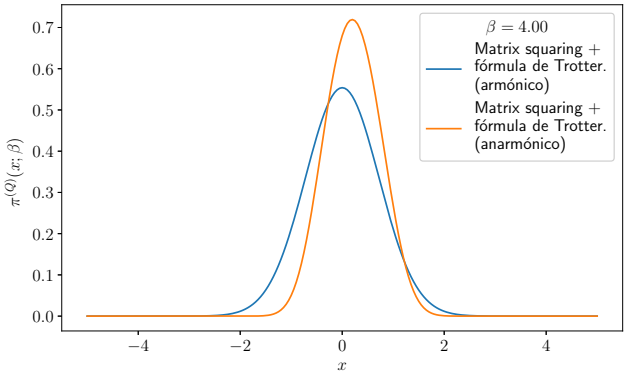

Figura 3. $\pi^{(Q)}_{\text{comp}}(x;\beta_{\text{fin}})$ para el potencial anarmónico (22). Se muestra también para comparación $\pi^{(Q)}_{\text{comp}}(x;\beta_{\text{fin}})$ del potencial armónico. Los parámetros del algoritmo fueron $\beta_{\text{fin}} = 4$, $\beta_{\text{ini}} = 2^{-9}\beta_{\text{fin}}$, $x_{\max} = 5$, $dx = 0.05$.

La figura 3 muestra el valor de $\pi^{(Q)}_{\text{comp}}(x;\beta_{\text{fin}})$ para el potencial anarmónico

$$V(x) = \frac{1}{2}x^2 - x^3 + x^4, \tag{22}$$

y se muestra también para comparación, el resultado para el potencial armónico. Notamos que la forma de general la curva es muy similar a la del oscilador anharmónico. Sin embargo, se nota la asimetría del potencial anarmónico, ya que hay un corrimiento hacia la derecha con respecto al armónico. También vemos que para el potencial anarmónico es más angosto el pico, esto se deduce ya

que su altura es máyor a comparación con el potencial armónico.

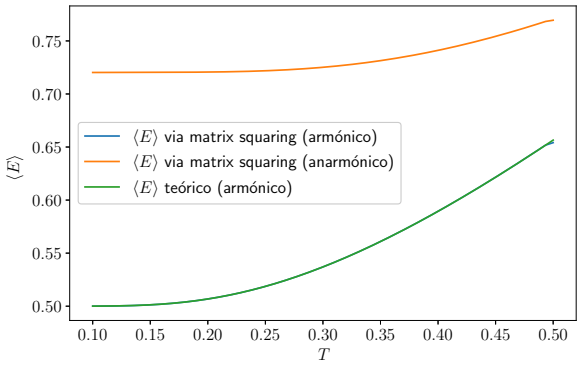

Figura 4. $\langle E\rangle$ dada por (25) y calculada usando las matrices obtenidas mediante algoritmo *Matrix Squaring* para los potenciales armónico y anharmónico. Los parámetros usados fueron $\beta_{\text{ini}} = 2^{-9}\beta_{\text{fin}}$, $x_{\max} = 5$, $dx = 0.05$. Se notan diferencias entre el valor teórico y el computacional para el potencial armónico pero solo en las puntas por el efecto de la derivada numérica en los extremos. Notamos que para el potencial armónico el límite de $T \to 0$ corresponde con el de la energía base $E = 1/2$, mientras que el caso del potencial anarmónico la energía en este límite es un poco myor. La forma de las curvas para el potencial anarmónico y el armónico son similares.

Con este mismo algoritmo es posible encontrar la función partición para diferentes valores de beta, ya que

$$Z(\beta) = \int dx \rho(x, x; \beta). \tag{23}$$

Con la ayuda de ésta se puede calcular a su vez el valor de la energía promedio del sistema mediante la conocida relación

$$\langle E\rangle = -\frac{\partial}{\partial\beta} \ln Z. \tag{24}$$

En la figura 4 se muestra la energía promedio para diferentes valores de temperatura cercanos a $T = 0$, tanto para el potencial armónico, como para el potencial anarmónico. Además, se muestra el valor teórico para el potencial armónico

$$\langle E\rangle = \frac{1}{2}\coth(\beta/2). \tag{25}$$

Encontramos que el calculo computacional de $\langle E\rangle$ para el potencial armónico está casi totalmente solapado con su contraparte teórica. Los errores apreciables están en las puntas de la gráfica y son debidos a efectos normales por cálculo numérico de la derivada en los extremos, que es más imprecisa que en la mitad de la gráfica. Notamos también que en el límite $T \to 0$, $\langle E\rangle \to 1/2$, tal y como se espera, ya que a temperatura cero el sistema tiende al estado base que en este caso tiene energía $E = 1/2$.

En el caso del cálculo computacional de $\langle E\rangle$ para el potencial anarmónico, vemos que el límite $T \to 0$ la energía es un poco mayor que para el caso del potencial armónico, es decir, se espera que la energía del estado base del oscilador anarmónico sea mayor y coincida con el límite que se muestra en esta gráfica. También notamos que en temperaturas cercanas a cero, la energía cambia más lentamente que cuando se compara con la gráfica del potencial armónico. Sin embargo, tal y como se espera, el comportamiento de $\langle E\rangle$ no es abruptamente diferente al del oscilador en este límite de 'bajas' temperatiuras.

Nota: los programas de Python3 en los que se implementó este algoritmo y con los que se realizaron estas gráficas se pueden encontrar en el apéndice A.

B. *Path Integral Naive Sampling*

Para este algoritmo, el camino que se cambiará en cada iteración consta de diez posiciones. Es decir, en (13) se toma $N = 10$. Además se tomó $\delta = 0.5$ y el número de iteraciones que se hicieron fue $N_{\text{iter}} = 10^6$.

En la figura 5 encontramos los resultados para el potencial armónico. En la figura a) vemos el histograma tomando solo las posiciones x_0 de X (13) que se guarda en cada iteración. Además podemos ver allí también el resultado del algoritmo *Matrix Squaring* que se muestra como línea continua (el eje y correspondiente a estos datos es el de la izquierda). Podemos ver que los resultados del algoritmo *Path Integral Naive Sampling* se ajustan en términos generales a la forma del resultado del algoritmo *Matrix Squaring* (que en este caso en términos prácticos es idéntico al teórico). Notamos sin embargo que hay bastantes fluctuaciones en los resultados del histograma, lo cual está muy relacionado con el hecho de que es un algoritmo tipo Montecarlo que depende de números aleatorios. Es probable que si se aumenta significativamente el número de iteraciones se logren disminuir las fluctuaciones. También esto puede estar relacionado con el número de *bins* que se usaron que fue $\sqrt{N_{\text{iter}}}$, que es una regla usada comunmente. Lo interesante de usar bastantes *bins* es que precisamente estas fluctuaciones son evidentes y notamos que el algoritmo no ha convergido totalmente al resultado esperado y que son necesarias más iteraciones para obtener punto a punto un resultado más cercano al teórico y no solo la forma general. Por el aumento de tiempo computacional no se intentó realizar un número significativamente mayor de iteraciones, pero en principio el algoritmo debe converger a la solución, tal como podemos intuir a partir del resultado obtenido con el valor de N_{iter} escogido. En esta misma figura a) encontramos también el camino $X = [x_0,\ x_1,\ \ldots,\ x_9]$ que resulta al final de las 10^6 iteraciones. Es interesante ver que los valores por los que pasa el camino se ubican predominantemente entre $x = -1$ y $x = 1$, esto en conformidad con la región de mayor probabilidad, de acuerdo con lo que vemos en el histograma. El eje y para este camino es el de la derecha, que marca los valores de β para cada paso del camino,

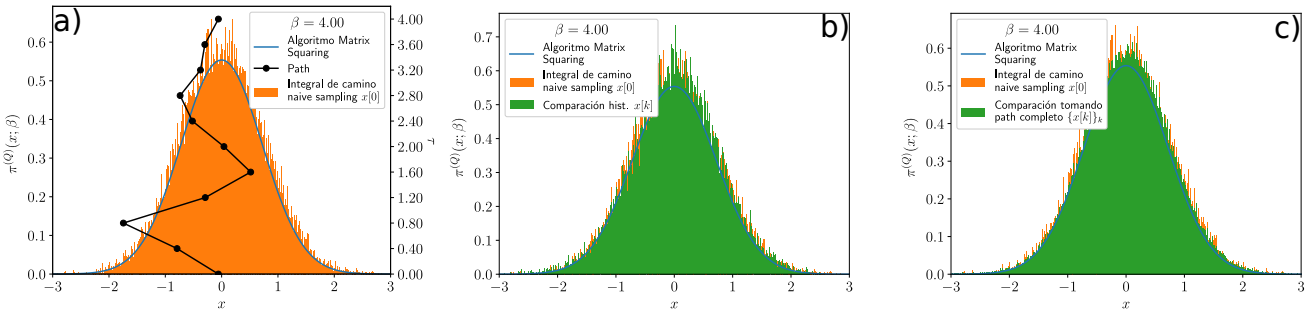

Figura 5. Histogramas para el potencial armónico obtenidas con el algoritmo *Path Integral Naive Sampling*. a) Histograma para posiciones x_0; semuestra también el path que se obtiene al final del algoritmo. b) Histograma para posiciones x_0 y comparación con histograma para x_k con $k \neq 0$ notamos que los histogramas coinciden en la forma y se ajustan al los valores dados por el algoritmo *Matrix Squaring*. c) Histograma para posiciones x_0 y comparación con histograma generado por los valores que toma todo el camino $[x_0,\ x_1,\ \ldots,\ x_9]$. Si se hace Zoom en éste último y se compara con los otros, podemos ver que las fluctuaciones alrededor del valor teórico disminuyen debido a que acá se consideran más datos. Los parámetros usados para este caso fueron $N_{\text{iter}} = 10^6$ y $\delta = 0.5$.

segun se acostumbra graficar.

En la figura 5c) podemos ver el histograma construido con la unión de todos los caminos por los que pasó el algoritmo y en la figura 5b) podemos ver el histograma construido con los valores de x_k con $k \neq 0$. En los dos casos vemos que el histograma coincide en términos generales con el mostrado en la figura 5a), que se muestra también en las figuras mencionadas para comparación. Hay un detalle sutil que hace diferencia en estos dos histogramas y es que cuando se construye con el camino completo (es decir, el que se muestra en 5c)) las fluctuaciones disminuyen. Esto se puede ver haciendo zoom en ambas figuras b) y c) y es debido a que en la figura c) por tratarse del camino completo, estamos teniendo en cuenta 10^7 posiciones y no 10^6 (el cual es el caso de las otras dos figuras). Al ser construido con más datos, las fluctuaciones alrededor del valor teórico disminuyen un poco.

Por otro lado en las figuras 6 encontramos las figuras análogas al caso anterior pero acá para el potencial anarmónico. Encontramos en términos generales que las características de estas figuras son muy similares a las del caso armónico. Las diferencias obvias son que este histograma está más corrido hacia la derecha, lo cual refleja el hecho de que este potencial es asimétrico con respecto al eje $x = 0$. Igualmente, en este caso se nota que las fluctuaciones disminuyen un poco cuando se considera el camino completo.

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, bajo un efoque computacional a la luz de los algoritmos *Matrix Squaring* y *Path Integral Naive Sampling*.

En el caso del primer algoritmo notamos que es robusto y los resultados para el potencial armónico coinciden con los valores teóricos. En este algoritmo pudimos implamentar una optimización para los parámetros β_{ini} y dx y encontramos que cuanto más pequeño sea β_{ini}, más pequeña debe ser la discretización dx para evitar errores computacionales. Se encontraron diferencias evidentes para $\pi^{(Q)}(x;\beta)$ entre el potencial armónico y el anarmónico, la más evidente es el desplazamiento del pico para el anarmónico hacia la derecha. Sin embargo, si solo se mira la forma de las curvas ambas son muy similares, aunque la del anarmónico tiene el pico con una densidad de probabilidad mayor. Acá también encontramos que los valores obtenidos para la energía promedio $\langle E \rangle$ coinciden con los teóricos y en el caso del oscilador armónico, el límite de baja temperatura coincide con la energía del estado base del sistema $E = 1/2$, mientras que para el potencial anarmónico la energía en este límite fue un poco más grande. Sin embargo, de nuevo, el comportamiento de la curva en su forma es similar para los dos casos.

En el caso del segundo algoritmo, encontramos que no es tan preciso como el primero pero es un método valioso para contrastar y corroborar los resultados que se obtuvieron. Encontramos que cuando se tiene en cuenta el camino completo para graficar el histograma, las fluctuaciones alrededor del valor teórico dismiuyeron un poco y esperamos que si se incrementa significativamante N_{iter}, estas fluctuaciones disminuyan aún más. Para los caminos graficados, tanto en el caso anarmónico como en el armónico, encontramos que los valores de las pociciones se concentran en las regiones de mayor probabilidad.

Para trabajos futuros sería interesante evaluar en más detalle el papel que juega el número de iteraciones en cada algoritmo así como el número de posiciones considerado para los caminos en el algoritmo de *Path Integral Naive Sampling*.

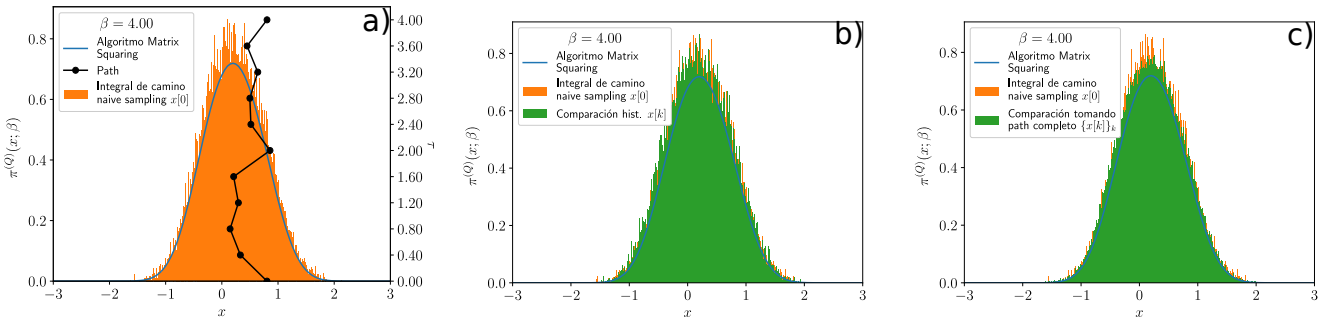

Figura 6. Histogramas para el potencial anararmónico (22) obtenidas con el algoritmo *Path Integral Naive Sampling*. a) Histograma para posiciones x_0; semuestra también el path que se obtiene al final del algoritmo. b) Histograma para posiciones x_0 y comparación con histograma para x_k con $k \neq 0$ notamos que los histogramas coinciden en la forma y se ajustan al los valores dados por el algoritmo *Matrix Squaring*. c) Histograma para posiciones x_0 y comparación con histograma generado por los valores que toma todo el camino $[x_0,\ x_1,\ \ldots,\ x_9]$. Si se hace Zoom en éste último y se compara con los otros, podemos ver que las fluctuaciones alrededor del valor teórico disminuyen debido a que acá se consideran más datos. Los parámetros usados para este caso fueron $N_{\text{iter}} = 10^6$ y $\delta = 0.5$.

Las implementaciones de los algoritmos usados en este trabajo 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).

[3] W. Krauth, *Statistical Mechanics: Algorithms and Computations*, 1st ed. (Oxford University Press, 2006).

[4] Richard P. Feynmann, *Statistical Mechanics: a Set of Lectures*, 2nd ed. (The Benjamin/Cummings Publishing Company., 1972) pp. 49–51.

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

[6] 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).

[7] M. E. J. Newmanand G. T. Barkema, *Monte Carlo Methods in Statistical Physics* (Oxford University Press, 1999) pp. 1–300.

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

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

Apéndice A: Código 1: Matrix Squaring

A continuación se muestra el código que corre los algoritmos que se usaron para la parte de Matrix Squaring en la sección III A. Éste código usa el módulo matrix_squaring que está disponible en este link. Cada función del módulo está comentada y explicada.

```
from matrix_squaring import *



##############################################################################################
# PANEL DE CONTROL
##############################################################################################

```

```
# Decide si corre algoritmo matrix squaring con aproximación de trotter
run_ms_algorithm = True

# Decide si corre algoritmo para cálculo de energía interna
run_avg_energy = True

# Decide si corre algoritmo para optimización de dx y beta_ini
run_optimization = False



#############################################################################################################
# PARÁMETROS GENERALES PARA LAS FIGURAS
#############################################################################################################

# Usar latex en texto de figuras y agrandar tamaño de fuente
plt.rc('text', usetex=True)
plt.rcParams.update({'font.size':15,'text.latex.unicode':True})

# Obtenemos path para guardar archivos en el mismo directorio donde se ubica el script
script_dir = os.path.dirname(os.path.abspath(__file__))



#############################################################################################################
# CORRE ALGORITMO MATRIX SQUARING
#############################################################################################################

if run_ms_algorithm:

    # Parámetros físicos del algoritmo
    physical_kwargs = {
        'beta_fin': 4,
        'x_max': 5.,
        'nx': 201,
        'N_iter': 9,
        'potential': harmonic_potential,
        'potential_string': 'harmonic_potential',
        }

    # Parámetros técnicos (generar archivos y figuras, etc.)
    technical_kwargs = {
        'print_steps': False,
        'save_data': True,
        'csv_file_name': None,
        'relevant_info': None,
        'plot': True,
        'plot_QHO_theory': True,
        'save_plot': True,
        'show_plot': False,
        'plot_file_name': None,
        }

    kwargs = {**physical_kwargs, **technical_kwargs}

    rho, trace_rho, grid_x = run_pi_x_sq_trotter(**kwargs)


```

```
67
68  ##########################################################################################################
69  # CORRE ALGORITMO PARA CÁLCULO DE ENERGÍA INTERNA
70  ##########################################################################################################
71
72  if run_avg_energy:
73
74      # Parámetros técnicos función partición y cálculo de energía
75      technical_Z_kwargs = {
76          'read_Z_data': False,
77          'generate_Z_data': True,
78          'Z_file_name': None,
79          'plot_energy': True,
80          'save_plot_E': True,
81          'show_plot_E': False,
82          'E_plot_name': None,
83          }
84
85      # Parámetros físicos para calcular Z y <E>
86      physical_kwargs = {
87          'temp_min': 1./10,
88          'temp_max': 1./2,
89          'N_temp': 300,
90          'potential': harmonic_potential,
91          'potential_string': 'harmonic_potential',
92          }
93
94      # Más parámetros técnicos
95      more_technical_kwargs = {
96          'save_Z_csv': True,
97          'relevant_info_Z': None,
98          'print_Z_data': False,
99          'x_max': 5.,
100         'nx': 201,
101         'N_iter': 9,
102         'print_steps': False,
103         'save_pi_x_data': False,
104         'pi_x_file_name': None,
105         'relevant_info_pi_x': None,
106         'plot_pi_x': False,
107         'save_plot_pi_x': False,
108         'show_plot_pi_x': False,
109         'plot_pi_x_file_name': None,
110         }
111
112     kwargs = {**technical_Z_kwargs, **physical_kwargs, **more_technical_kwargs}
113
114     average_energy(**kwargs)
115
116
117
118 ##########################################################################################################
119 # CORRE ALGORITMO PARA OPTIMIZACIÓN DE DX Y BETA_INI
120 ##########################################################################################################
121
122 if run_optimization:
123
124     # Parámetros físicos
```

```
    physical_kwargs = {
        'beta_fin': 4,
        'x_max': 5,
        'potential': harmonic_potential,
        'potential_string': 'harmonic_potential',
        'nx_min': 20,
        'nx_max': 1121,
        'nx_sampling': 50,
        'N_iter_min': 9,
        'N_iter_max': 20,
        }

    # Parámetros técnicos
    technical_kwargs = {
        'generate_opt_data': True,
        'read_opt_data': False,
        'save_opt_data': True,
        'opt_data_file_name': None,
        'opt_relevant_info': None,
        'plot_opt': True,
        'show_plot_opt': True,
        'save_plot_opt': True,
        'opt_plot_file_name': None,
        'print_summary': True,
        }

    kwargs = {**physical_kwargs, **technical_kwargs}

    error, dx_grid, beta_ini_grid, comp_time = optimization(**kwargs)
```

Apéndice B: Código 2: Naive Path Integral Montecarlo Sampling

A continuación se muestra el código que corre los algoritmos que se usaron para la parte de *Path Integral Naive Sampling* en la sección III B. Éste código usa el módulo path_integral_naive_sampling que está disponible en este link. Cada función del módulo está comentada y explicada.

```
from path_integral_naive_sampling import *



##########################################################################################################
# PANEL DE CONTROL
##########################################################################################################

# Decide si corre algoritmo path integral naive sampling (genera paths y guarda en CSV)
run_path_int = True

# Decide si corre algoritmo para generar figura con histograma x[0] y un camino aleatorio.
# Además muestra comparación con resultado de algoritmo matrix squaring.
run_plot_1 = True

# Decide si corre algoritmo para generar figura con histograma x[0] e histograma x[k].
# Además muestra comparación con resultado de algoritmo matrix squaring.
run_plot_2 = True

# Decide si corre algoritmo para generar figura con histograma x[0] e histograma
# completo {x[k]}. Además muestra comparación con resultado de algoritmo matrix squaring.
```

```python
run_plot_3 = True



##########################################################################################################
# PARÁMETROS GENERALES PARA LAS FIGURAS
##########################################################################################################

# Usar latex en texto de figuras y agrandar tamaño de fuente
plt.rc('text', usetex=True)
plt.rcParams.update({'font.size':15,'text.latex.unicode':True})
# Obtenemos path para guardar archivos en el mismo directorio donde se ubica el script
script_dir = os.path.dirname(os.path.abspath(__file__))

potentials = [
              [anharmonic_potential,
               'anharmonic_potential',
               'pi_x-ms-anharmonic_potential-beta_fin_4.000-x_max_5.000-nx_201-N_iter_9.csv'],
              [harmonic_potential,
               'harmonic_potential',
               'pi_x-ms-harmonic_potential-beta_fin_4.000-x_max_5.000-nx_201-N_iter_9.csv']
             ]

for i, potential in enumerate():


    ##########################################################################################################
    # Corre algoritmo y guarda caminos en lista pathss_x Además guarda datos en archivo CSV
    ##########################################################################################################

    # Parámetros físicos
    kwargs = {
        'N_path': 10,
        'beta': 4.,
        'N_iter': int(1e6),
        'delta': 0.5,
        'potential': potential[0],  # harmonic_potential,
        'potential_string': potential[1],  # 'harmonic_potential',
        'append_every': 1,
        'save_paths_data': True,
        'paths_file_name': None,
        'paths_relevant_info': None,
        'pathss_x': None,
        }

    # Calcula caminos de integral de camino pathss_x = [camino_1, camino_2, ...]
    # camino_i = [x_i[0], x_i[1], x_i[2], ..., x_i[N_path-1]]
    if run_path_int:
        pathss_x = path_naive_sampling(**kwargs)



    ##########################################################################################################
    # Primera figura: muestra histograma x[0] y un path.
    ##########################################################################################################

    # Parámetros técnicos de figuras
    kwargs_update = {
```

```
        'pathss_x': pathss_x,
        'read_paths_data': False,
        'paths_file_name': None,
        'N_plot': 201,
        'x_max': 3,
        'N_beta_ticks': kwargs['N_path'] + 1,
        'msq_file': potential[2], #'pi_x-ms-harmonic_potential-beta_fin_4.000' #+
        ↪  '-x_max_5.000-nx_201-N_iter_7.csv',
        'plot_file_name': None,
        'show_QHO_theory': False,
        'show_matrix_squaring': True,
        'show_path': True,
        'show_compare_hist': False,
        'show_complete_path_hist': False,
        'save_plot': True,
        'show_plot': False,
        }

    kwargs.update(kwargs_update)

    if run_plot_1:
        figures_fn(**kwargs)



    ###########################################################################################
    # Segunda figura: compara histograma x[0] con histograma hecho con x[0],...,x[N-1]
    ###########################################################################################

    # Parámetros técnicos de figuras
    kwargs_update = {
        'show_path': False,
        'show_compare_hist': True,
        'save_plot': True,
        'show_plot': False,
        }

    kwargs.update(kwargs_update)

    if run_plot_2:
        figures_fn(**kwargs)



    ###########################################################################################
    # Tercera figura: compara histograma x[0] con histograma hecho con x[0],...,x[N-1]
    ###########################################################################################

    # Parámetros técnicos de figuras
    kwargs_update = {
        'show_compare_hist': False,
        'show_complete_path_hist': True,
        'save_plot': True,
        'show_plot': False,
        }

    kwargs.update(kwargs_update)

```

```
137     if run_plot_3:
138         figures_fn(**kwargs)
```