Estimación de estados cuánticos simétricos generados por computadoras cuánticas con ruido

Giannina Zerr[1], Federico Holik[2] y Marcelo Losada[3]

[1] Departamento de Fısica, FCEyN, Universidad de Buenos Aires,
1428 Buenos Aires, Argentina,
giannina.zerr@gmail.com
[2] Instituto de Fısica La Plata (CONICET),
Boulevard 113 & 63, La Plata, C.P. 1900, Argentina
[3] Facultad de Matemática, Astronomıa, Fısica y Computación,
Universidad Nacional de Córdoba - CONICET

Resumen En este proyecto abordamos el problema de reconstrucción de estados cuánticos simétricos utilizando la técnica de estimación denominada **Group Invariant Quantum Tomography (GIT)**, la cual permite una disminuación de las mediciones necesarias. En este trabajo presentamos dos códigos [1] desarrollados en Python, utilizando la librería Amazon Braket. El primero realiza la simulación de circuitos cuánticos y mediciones, considerando el sistema bajo la presencia de ruido. El segundo código implementa las tomografías VQT-GIT basado en optimización convexa, que reconstruye la matriz densidad del estado a partir de los datos obtenidos en las mediciones. Estos códigos fueron empleados en el trabajo [2] para evaluar la fidelidad del método tomográfico. Los resultados muestran que, utilizando un número considerablemente menor de mediciones en comparación con la tomografía convencional, es posible obtener reconstrucciones de alta fidelidad.

Keywords: tomografía cuántica, VQT, GIT

1. Introducción

Una variante del método de estimación de Máxima Entropía (MaxEnt) es la llamada **Variational Quantum state Tomography** (VQT) [3]. Consiste en resolver el siguiente problema de optimización convexo:

$$\begin{aligned} \min_{\rho,\,\Delta} \left(\alpha \sum_{i\in\mathcal{I}} \Delta_i + \beta \sum_{i\notin\mathcal{I}} \mathrm{tr}(E_i\rho) - \gamma \log\left(\det(\rho)\right) \right), \\ \text{sujeto a } |\mathrm{tr}(E_i\rho) - f_i| \leq \Delta_i f_i \;\; i \in \mathcal{I}, \\ \Delta_i \geq 0, \\ \mathrm{tr}(\rho) = 1, \\ \rho \succeq 0, \end{aligned} \tag{1}$$

donde $\{E_i\}$ es el conjunto de medidas, $\{f_i\}$ las frecuencias medidas (o valores medios), $\{\Delta_i\}$ las tolerancias, e I representa el conjunto de índices de los datos medidos. La función $\log(\det(\rho))$ se denomina función barrera, ya que si se intenta salir del conjunto $\rho > 0$, se llega a un punto donde al menos uno de los autovalores desaparece [4]; los parámetros α, β y γ actúan como factores de escala.

Este método tomográfico puede combinarse con conocimiento previo acerca de las simetrías, dado que, para ciertos grupos de simetría, es posible utilizar un menor número de parámetros para especificar un estado arbitrario. En dicho caso, el método anterior puede ser llamado **Group Invariant Quantum Tomography** (GIT) [5]. Como consecuencia, es posible obtener una reducción considerable del costo computacional del proceso y, a su vez, una reducción en los costos experimentales. Este método puede aplicarse a simetrías arbitrarias y generaliza los planteamientos anteriores [6].

Con la motivación de estudiar la eficiencia del método GIT se procedió a crear dos códigos complementarios, estos se encuentran disponibles en el repositorio de GitHub de la referencia [1]. Los códigos fueron desarrollados en el marco de la realización de tesis de Licenciatura de Giannina Zerr, bajo supervisión de Federico Holik como director y Marcelo Losada como co-director. A su vez, los códigos fueron utilizados en el trabajo de investigación *"Group-invariant estimation of symmetric states generated by noisy quantum computers"* realizado por F. Holik, M. Losada, G. Zerr, L. Rebón y D. Tielas [2].

El objetivo del primer código *"Measurements.py"* es simular ciertos estados cuánticos simétricos con ruido y mediciones realizadas sobre estos, para lo cual se utilizó la librería Amazon Braket [7] de Python. Las mediciones obtenidas son guardadas en la carpeta *"Measurements"* (si no existe, el código la crea en la primera corrida).

Luego, el segundo código *"Tomography.py"* lee las mediciones obtenidas para realizar el método GIT. Las estimaciones de la matriz densidad obtenidas con GIT son comparadas con el estado original simulado para obtener la eficiencia del método, en función del número de shots empleados y el nivel de ruido. En el mismo script se encuentran varios graficadores para visualizar mejor los resultados. Todo el material generado es guardado en la carpeta *"Results"* (si no existe, el código la crea en la primera corrida).

En las secciones 2 y 3 se ampliará el detalle de las funciones desarrolladas para cada código.

Nota: El archivo *"2in1_Measurements_and_Tomography.py"* contiene ambos códigos en caso de querer correrlos conjuntamente.

2. Circuitos y Mediciones

Este código tiene como objetivo simular estados cuánticos mediante circuitos utilizando el simulador de Amazon Braket, y realizar mediciones sobre dichos estados en distintas bases. El código permite generar estados tipo GHZ de cualquier número de qubits o de Werner de 2 qubits, introducir distintos tipos y

niveles de ruido, y realizar mediciones sobre distintas bases, incluyendo simetría completa o permutacional.

2.1. Elección de entradas

El código contiene diversas funciones para automatizar las pruebas a realizar. En principio, permite seleccionar las siguientes opciones:

- **Estado**: puede ser GHZ de cualquier número de qubits o Werner de 2 qubits con elección del parámetro p entre 0 y 1, descripto por la ecuación (2) (ver sección 2.2).
- **Simetría**: puede ser completa o permutacional. En el caso de seleccionar la simetría completa se generan todos los productos tensoriales posibles entre estos operadores, incluyendo permutaciones. En el caso de seleccionar simetría permutacional se obtienen únicamente combinaciones con reemplazo que son invariantes bajo permutaciones de los qubits (ver sección 2.3).
- **Dispositivo**: la opción actual por defecto es LocalSimulator("braket_dm")
- **Número de shots**: la medición de un sistema cuántico es intrínsecamente probabilística, el resultado de una única medición dará la proyección sobre uno de los posibles estados. Por este motivo es necesario repetir las mediciones varias veces. En la jerga de computación cuántica, a cada repetición se le dice *shot*. El código permite repetir el proceso de medición para varios valores de número de shots, por lo que es posible seleccionar los valores deseados expresados en forma de lista.
- **Número de mediciones**: dado un valor fijo de número de shots, las estimaciones de estados deben repetirse una cierta cantidad de veces para, finalmente, obtener un valor de fidelidad promedio de los experimentos realizados.
- **Ruido**: los tipos de ruido que incluye el modelo de ruido de nuestro código son Depolarizing, BitF Flip, Amplitude Damping; puede seleccionarse una lista ya que el código está automatizado para repetir el proceso para varios tipos de ruidos. A su vez, deben elegirse los niveles de ruido deseados en forma de lista (ver sección 2.4).

2.2. Generación de estados

Un **estado GHZ** de N qubits es de la forma $|\text{GHZ}\rangle = \frac{1}{\sqrt{2}} |0\rangle^{\otimes M} + |1\rangle^{\otimes M}$. La implementación en circuito de un GHZ puede verse en la figura 1.

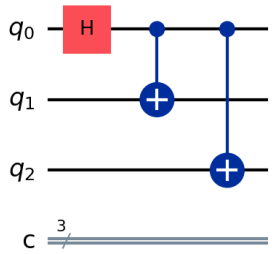

Figura 1. Implementación del circuito para un estado GHZ de 3 qubits.

Dado p un parámetro real entre 0 y 1, la matriz densidad de un **estado de Werner** de dos qubits puede escribirse como

$$\rho_p = \left(\frac{1-p}{4}\right) I + p|\psi^-\rangle\langle\psi^-| \qquad \text{con} \quad |\psi^-\rangle = \frac{1}{\sqrt{2}}(|01\rangle - |10\rangle) \qquad (2)$$

La implementación del circuito de Werner de 2 qubits puede verse en la figura 2, donde los ángulos θ_i están definidos por el parámetro p de la ecuación (2). [8] Dicha relación se encuentra dentro del código entre el array ps=np.linspace(0,1,50) y el array denominado *thetas*, donde el valor de p dado en una cierta posición en el array *ps* se corresponde con los valores de θ dados en la misma posición del array *thetas*.

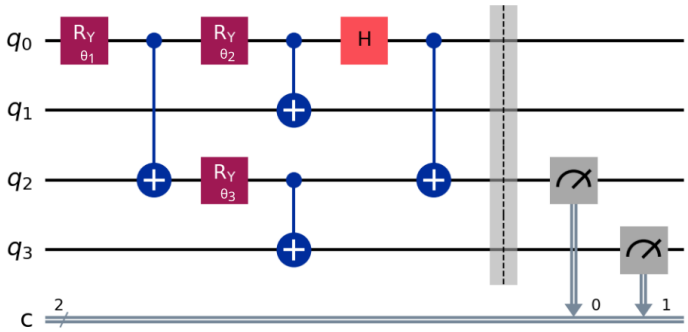

Figura 2. Implementación del circuito para un estado de Werner de 2 qubits. Los qubits sobre los que se mide son el q_0 y q_1. Los valores θ_1, θ_2 y θ_3 dependen del parámetro p elegido (correspondiente a la ecuación (2)).

2.3. Generación de observables

En la computación cuántica, una medición sobre un qubit será por defecto en la base computacional, es decir el operador de Pauli σ_z. Para realizar mediciones

sobre otras bases se deben aplicar las correspondientes compuertas cuánticas. En la figura 3 se observan las mediciones sobre las distintas bases de las matrices de Pauli, X, Y y Z. [9]

En nuestro código hemos definido la función *local_measurement(N)* que genera los circuitos de medición de las 3 bases para el número de qubits seleccionado.

Para generar la lista de observables sobre la cual se realizará las mediciones, se implementó la función *"observables(N, symmetry)"*. La misma utiliza la librería *intertools* que permite generar todas las combinaciones posibles del producto tensorial de los operadores I, X, Y y Z dependiendo del número de qubits seleccionado. En el caso de seleccionar "symmetry" como "Complete" se generan todos los productos tensoriales posibles entre estos operadores, incluyendo permutaciones. En el caso de seleccionar "Permutational" se obtienen únicamente combinaciones con reemplazo que son invariantes bajo permutaciones de los qubits. Por ejemplo para 2 qubits, tenemos que la base de medición completa posee 16 observables: II, IX, IY, IZ, XI, XX, XY, XZ, YI, YX, YY, YZ, ZI, ZX, ZY, ZZ, pero si consideramos simetría permutacional la base se reduce a 10 observables: II, IX, IY, IZ, XX, XY, XZ, YY, YZ, ZZ.

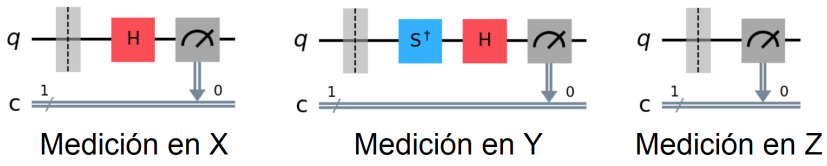

Figura 3. Mediciones sobre las bases X, Y y Z.

2.4. Modelo de ruido

Las computadoras cuánticas son susceptibles a efectos del entorno, lo que produce que el rendimiento de las operaciones no sea el esperado. A su vez, los qubits pueden sufrir errores que no los hagan cambiar en las cantidades deseadas. Por estos motivos, el estudio del ruido es crucial en el área de la computación cuántica, en pos de construir sistemas de procesamiento cuántico robustos. Algunos de los principales ruidos son

- **Depolarizing:** un qubit es reemplazado por un estado completamente mixto de la base computacional.
- **Bitflip:** un qubit en estado $|0\rangle$ cambia a un estado $|1\rangle$ (y viceversa).
- **Amplitude Damping:** se basa en la disipación de energía, donde un qubit en estado $|1\rangle$ decae a un estado $|0\rangle$ de forma irreversible.

En el presente trabajo se estudió el efecto en los circuitos cuánticos de estos tres tipos de ruido. Para lo mismo se utilizó el modelo de ruido incluido en la

librería de Amazon Braket. El mismo utiliza la modelación a través de la descomposición de Kraus, la cual es una representación matemática de los canales cuánticos [10].

En el presente trabajo implementamos la función *noise(x)* para aplicar el modelo de ruido presente en Braket aplicado a compuertas cuánticas. El modelo afecta a las compuertas cuánticas H, $CNOT$ y R_y con tres tipos posibles de ruido: depolarizing, bit flip y amplitud damping. Al valor de probabilidad p lo llamaremos "nivel de ruido", los valores utilizados fueron entre 0 y 0,15 para compuertas H y R_y, y entre 0 y 0,015 para compuertas CNOT. El motivo por el cual el nivel de ruido aplicado a compuertas $CNOT$ es menor es debido a que las compuertas aplicadas a dos qubits generan mayor cantidad de error que las aplicadas a un solo qubit para un mismo valor de p.

Con el objetivo de estudiar el efecto de cada tipo de ruido por separado, hemos definido la función *noisy(noise_type)*. La misma produce un diccionario para aplicar los niveles de ruido elegidos al tipo de ruido seleccionado, mientras que los otros tipos de ruido no son aplicados.

2.5. Mediciones

La función *measurements(circuit, L0, noise_level, shot, device, target)* permite repetir una serie de mediciones variando la base sobre la cual se mide y el nivel de ruido, para un valor fijo de número de shots. *Circuit* será el circuito generado del estado que se desea estudiar, *L0* es la lista de observables generada por la función *local_measurement(N)* (según la simetría seleccionada), *shot* un valor fijo de número de shots a aplicar, y *device* el dispositivo cuántico seleccionado. El valor *Target* corresponde a los qubits sobre los cuales se realiza la medición (como vimos anteriormente, para GHZ las mediciones deben realizarse sobre todos los qubits del circuito, pero para Werner sólo sobre los qubits 3 y 4).

Como salida obtenemos el diccionario *Tomos_Noises*, donde las keys serán los observables medidos, y los valores asociados serán los valores medios de las mediciones realizadas para cada nivel de ruido. También obtenemos a la salida *DM*, que será la matriz densidad del estado generado con aplicación de ruido.

Finalmente, el código aplica esta función *measurements* en un bucle for para los distintos tipos de ruidos seleccionados y para distintos valores de shots. Los resultados obtenidos se guardan en la carpeta *Measurements*, donde también se guardan las entradas seleccionadas y la matriz densidad del estado sin ruido como *DM_ideal*.

3. Tomografía

Este código tiene como propósito reconstruir estados cuánticos utilizando técnicas de tomografía cuántica para obtener la matriz densidad del sistema. En principio, el mismo está diseñado para aplicar el método sobre las mediciones obtenidas en las simulaciones anteriores, es decir, la selección de entradas al

principio del código indicará de qué carpeta hacer la lectura de las mediciones obtenidas.

Como se explicó en la sección 1, el método tomográfico utilizado está dado por la ecuación (1) junto a sus respectivas restricciones. El mismo consiste en una optimización convexa, para lo cual se utilizó la librería CVXPY de Python. Se definen como variables los coeficientes α_i asociados a cada elemento S_i de la base para describir la matriz densidad ρ como combinación lineal de estos, es decir $\rho = \sum_{i=1}^{r} \alpha_i S_i$. Como vimos en la sección 1, el número de parámetros necesarios r para describir la matriz densidad se reduce para determinadas simetrías. Luego, estos coeficientes serán determinados por la optimización realizada.

Las bases ortonormales sobre las que se realiza la reconstrucción tomográfica están previamente generadas, se encuentran en la carpeta *Orthogonal_Basis*, y dependen de la simetría (completa o permutacional).

Para la resolución de la optimización, también son definidas como variables las tolerancias, es decir, el valor Δ_i de la ecuación (1).

Una vez reconstruida la matriz densidad, se calcula la fidelidad del método comparando el estado obtenido por la tomografía con el estado original generado por la simulación. La fidelidad es calculada utilizando una función auxiliar *SP_functions.py*, que se halla en el repositorio. Los datos obtenidos son guardados en la carpeta *"Results"*.

3.1. Visualización de resultados

El código contiene varios graficadores para visualizar los resultados. Para cada tipo de ruido se generan los gráficos de fidelidad en función del nivel de ruido, dentro del mismo bucle for en el que se realiza la tomografía. Al final del código se realizan gráficos comparativos de todos los tipos de ruido, de fidelidad en función del número de shots para un valor de nivel de ruido fijo, y de fidelidad en función del nivel de ruido para un número fijo de shots. En el segundo caso, se calcula la fidelidad al comparar: (1) el estado obtenido por tomografía con el estado ideal de la simulación, es decir, sin ruido. (2) el estado obtenido por tomografía con el estado de la simulación luego de la aplicación del modelo de ruido.

Es esperable que para el caso (1) el estado obtenido se aleje del estado ideal sin ruido a medida que se aumenta el nivel de ruido aplicado. Por el contrario, en el caso (2) se espera que el estado obtenido con el generado sean similares, para demostrar la efectividad del método. En la figura 4 se observan los gráficos comparativos generados, donde se observa esta tendencia. Los resultados obtenidos de esta investigación, publicados en la referencia [2], demuestran que la tomografía GIT posee un alto grado de fidelidad tal como sucede con el método VQT completo, sólo que en el caso de GIT se llega a resultados similares utilizando menos recursos al utilizar el conocimiento sobre las simetrías.

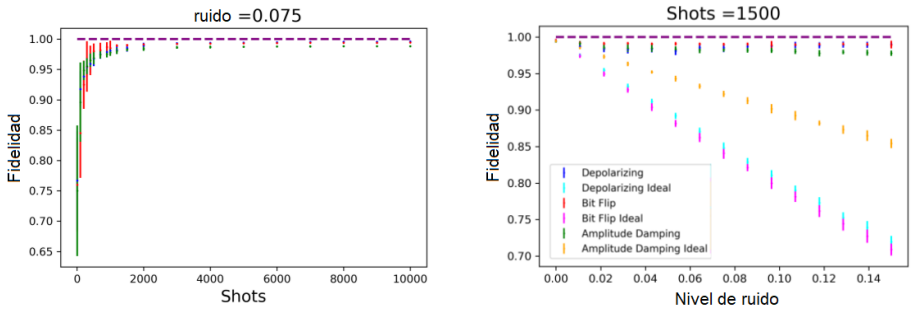

Figura 4. Fidelidades de los métodos completos de estimación GIT para las estadísticas simuladas de un estado GHZ de tres qubits con respecto al objetivo y al estado real. Cada tomografía se repite 30 veces (barra de error).

4. Conclusiones

Este código permite realizar reconstrucciones tomográficas de estados cuánticos simulados bajo distintos tipos de ruido y configuraciones experimentales. Su diseño modular y uso de optimización convexa lo convierte en una herramienta útil para estudios de robustez y caracterización de sistemas cuánticos.

Los resultados obtenidos de la implementación de este trabajo se encuentran publicados en nuestro trabajo de la referencia [2]. La investigación ha demostrado que la estimación GIT está en buen acuerdo con la estimación VQT completa, y permite una reducción sustancial de los recursos (tanto experimentales como computacionales). La GIT puede utilizarse como método rápido y barato para evaluar si un dispositivo cuántico es capaz de generar estados simétricos. Dado que el conjunto de estados simétricos con diferentes simetrías es muy rico en recursos cuánticos (entrelazamiento, no localidad y contextualidad, como es el caso de los estados GHZ), esta metodología puede utilizarse para evaluar el dispositivo cuántico en su conjunto.

Referencias

[1] Giannina Zerr, Federico Holik y Marcelo Losada. *Quantum Tomography for Symmetric States*. 2025. URL: https://github.com/gianninazerr/Quantum-Tomography-for-Symmetric-States.

[2] Federico Holik et al. "Group-invariant estimation of symmetric states generated by noisy quantum computers". En: *arXiv* (August 17 2024).

[3] D. S. Gonçalves et al. "Quantum state tomography with incomplete data: Maximum entropy and variational quantum tomography". En: *Phys. Rev. A* 87 (5 mayo de 2013), pág. 052140.

[4] Moroder,T. "Permutationally invariant state reconstruction". En: *New J. Phys.* 14 (2012), pág. 105001.

[5] Inés Corte et al. “Parameterizing density operators with arbitrary symmetries to gain advantage in quantum state estimation”. En: *arXiv* (2022). URL: https://arxiv.org/abs/2208.06540.

[6] G. Tóth et al. “Permutationally Invariant Quantum Tomography”. En: *Phys. Rev. Lett.* 105 (25 dic. de 2010), pág. 250403. URL: https://link.aps.org/doi/10.1103/PhysRevLett.105.250403.

[7] Amazon Web Services. *Amazon Braket*. 2020.

[8] Elias Riedel Gårding et al. “Bell Diagonal and Werner State Generation: Entanglement, Non-Locality, Steering and Discord on the IBM Quantum Computer”. En: *Entropy* 23.7 (2021). ISSN: 1099-4300. DOI: 10.3390/e23070797. URL: https://www.mdpi.com/1099-4300/23/7/797.

[9] Microsoft. *Single and multi-qubit Pauli measurement operations*. https://learn.microsoft.com/en-us/azure/quantum/concepts-pauli-measurements. 2024.

[10] J. Moreno, G. y Salton y T. Chen. *Noise in Quantum Computing*. https://aws.amazon.com/es/blogs/quantum-computing/noise-in-quantum-computing/. 2022.