First studies of Block encoding

Shi Qiu

Motivation

- The combinatorial track-finding problem can be mapped to solving the system of linear equation:

$$Ax = b \Rightarrow x = A^{-1}b$$

- The linear algebraic structure is compatible with algorithms like HHL, which has runtime scales like

$$O(\log N\, s^2\kappa^2/\epsilon)$$

where s is sparsity, κ is the condition number ($\lambda_{\max}/\lambda_{\min}$), and ϵ is the target error in the output state ($\| |\tilde{x}\rangle - |x^\star\rangle \|$).

- QSVT says that if $\boldsymbol{A}/\boldsymbol{\alpha_A}$ (α_A subnormalization factor) is **block encoded** by a (big) oracle O, then one can **block-encode** a scaled version of $\boldsymbol{A^{-1}}$ using about

$$O(\kappa \log(1/\epsilon))$$

Block encode a general matrix

- Block encoding means that the target matrix A (properly scaled by α_A) is placed in the upper-left corner of the larger unitary.

$$U = \begin{bmatrix} A/\alpha & \cdot \\ \cdot & \cdot \end{bmatrix} \qquad \Longrightarrow \qquad A = \alpha(\langle 0| \otimes I)U(|0\rangle \otimes I)$$

- General idea is that you apply the rotation to ancilla qubit(s) where the rotation angle is related to each entry values.
- Three registers:
  - ancilla: 1 qubit, where amplitudes get loaded
  - i register: n qubits (the “work” register representing rows),
  - j register: n qubits (the “data” register representing columns / input basis states)

Block encode a general matrix

- Starting with an arbitrary input $|\Psi\rangle = \sum_{j=0}^{N-1} \psi_j |j\rangle$, where $N = 2^n$ is the matrix size of A

$$|\Psi_0\rangle = |0\rangle_a |0^n\rangle_W \otimes |\Psi\rangle_J$$

- D_s apply Hadamards on $|0^n\rangle_W$:

$$|\Psi_0\rangle \xrightarrow{H^{\otimes n}|0^n\rangle_W} \frac{1}{\sqrt{N}} \sum_{\ell,j} \psi_j \, |0\rangle_a |\ell\rangle_W |j\rangle_J$$

- O_A rotates the amplitude ancilla conditioned on (ℓ, j) so that:

$$|\Psi_1\rangle \xrightarrow{O_A} |\Psi_2\rangle = \frac{1}{\sqrt{N}} \sum_{\ell,j} \psi_j \left(A_{\ell,j} |0\rangle_a + \sqrt{1 - A_{\ell,j}^2} |1\rangle_a \right) |\ell\rangle_W |j\rangle_J$$

- SWAP the two n-qubit registers and does not touch ancilla qubit

$$|\Psi_2\rangle \xrightarrow{\text{SWAP}} |\Psi_3\rangle = \frac{1}{\sqrt{N}} \sum_{\ell,j} \psi_j \left(A_{\ell,j} |0\rangle_a + \cdots |1\rangle_a \right) |j\rangle_W |\ell\rangle_J$$

- $D_s^\dagger$ applys final $H^{\otimes n}$ on the work register ($D_s^\dagger |j\rangle = \frac{1}{\sqrt{N}} \sum_{t=0} (-1)^{t \cdot j} |t\rangle$, where $t \cdot j$ is the bitwise inner product mod 2

$$|\Psi_3\rangle = \frac{1}{N} \sum_t \sum_{\ell,j} \psi_j (-1)^{t \cdot j} \left(A_{\ell,j} |0\rangle_a + \cdots |1\rangle_a \right) |t\rangle_W |\ell\rangle_J$$

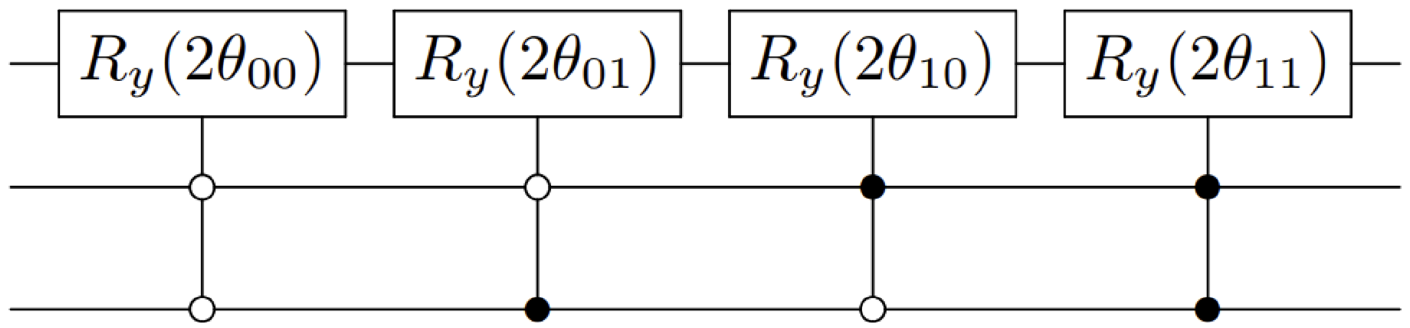
FABLE method to block encode a general matrix

- Goal: Implement O_A in simple 1- and 2-qubit gates for arbitrary matrices
- O_A acts on the $|0\rangle_a$ state as an R_y gate with angle

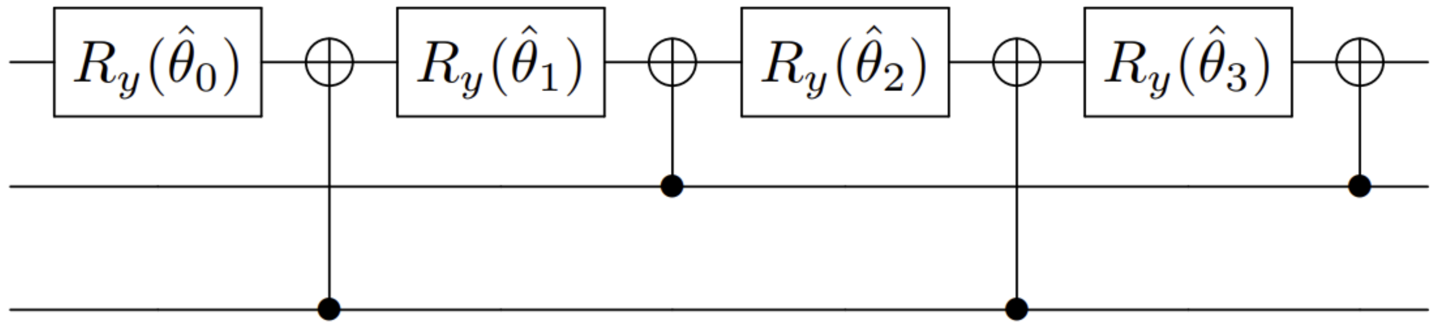
$$\theta_{ij} = \arccos(A_{ij})$$

- Naïve implementation of O_A will use N^2 multi-controlled Ry gates

  e.g. for a 2×2 matrix



- FABLE method implement O_A following gray code order (00, 01, 11, 10)



	Gray code			
	4	3	2	1
0	0	0	0	0
1	0	0	0	1
2	0	0	1	1
3	0	0	1	0
4	0	1	1	0
5	0	1	1	1
6	0	1	0	1
7	0	1	0	0
8	1	1	0	0
9	1	1	0	1
10	1	1	1	1
11	1	1	1	0
12	1	0	1	0

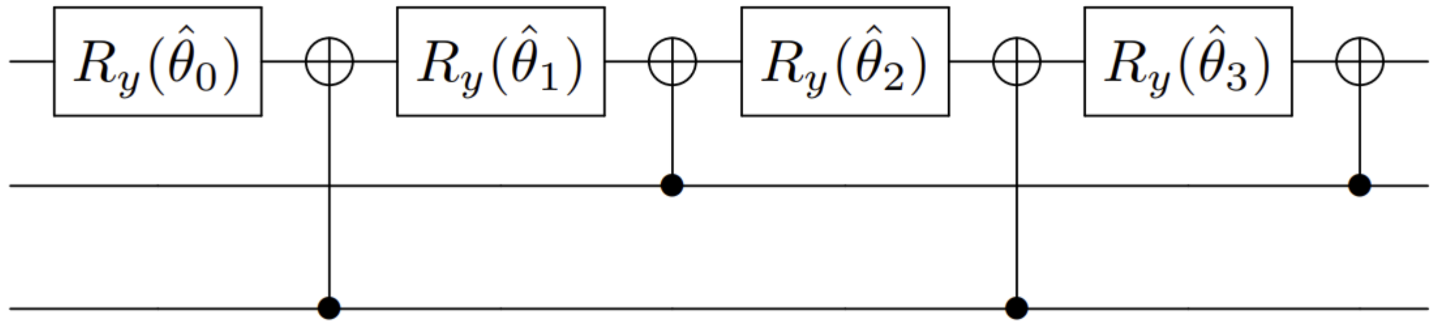
FABLE: Fast Approximate Quantum Circuits for Block-Encodings

https://arxiv.org/pdf/2205.00081

FABLE method to block encode a general matrix

- FABLE method uses the key identity: $XR_y(\theta)X = R_y(-\theta)$
- The state of the first qubit is rotated as:

$$
\begin{aligned}
00: \quad & R_y(\hat{\theta}_3) \quad R_y(\hat{\theta}_2) \quad R_y(\hat{\theta}_1) \quad R_y(\hat{\theta}_0) = \\
& \qquad R_y(\ \ \hat{\theta}_3 + \hat{\theta}_2 + \hat{\theta}_1 + \hat{\theta}_0), \\
01: \quad & R_y(\hat{\theta}_3) X R_y(\hat{\theta}_2) \quad R_y(\hat{\theta}_1) X R_y(\hat{\theta}_0) = \\
& \qquad R_y(\ \ \hat{\theta}_3 - \hat{\theta}_2 - \hat{\theta}_1 + \hat{\theta}_0), \\
10: \quad & X R_y(\hat{\theta}_3) \quad R_y(\hat{\theta}_2) X R_y(\hat{\theta}_1) \quad R_y(\hat{\theta}_0) = \\
& \qquad R_y(-\hat{\theta}_3 - \hat{\theta}_2 + \hat{\theta}_1 + \hat{\theta}_0), \\
11: \quad & X R_y(\hat{\theta}_3) X R_y(\hat{\theta}_2) X R_y(\hat{\theta}_1) X R_y(\hat{\theta}_0) = \\
& \qquad R_y(-\hat{\theta}_3 + \hat{\theta}_2 - \hat{\theta}_1 + \hat{\theta}_0),
\end{aligned}
$$



- The new angles $\hat{\theta}$ are related to θ as:

$$
\begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & -1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \begin{bmatrix} \hat{\theta}_0 \\ \hat{\theta}_1 \\ \hat{\theta}_2 \\ \hat{\theta}_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 0 & 1 \\ & & 1 & 0 \end{bmatrix} \begin{bmatrix} \hat{\theta}_0 \\ \hat{\theta}_1 \\ \hat{\theta}_2 \\ \hat{\theta}_3 \end{bmatrix} = (\hat{H} \otimes \hat{H}) P_G \begin{bmatrix} \hat{\theta}_0 \\ \hat{\theta}_1 \\ \hat{\theta}_2 \\ \hat{\theta}_3 \end{bmatrix}
$$

FABLE method to block encode a general matrix

- For an O_A oracle with angles $\boldsymbol{\theta} = (\theta_0, \ldots, \theta_{2^{2n}-1})$, the angles $\widehat{\boldsymbol{\theta}} = (\hat{\theta}_0, \ldots, \hat{\theta}_{2^{2n}-1})$ is related to $\boldsymbol{\theta}$ through:

$$\left(H^{\otimes 2n} P_G\right)\widehat{\boldsymbol{\theta}} = \boldsymbol{\theta} \quad \Rightarrow \quad \widehat{\boldsymbol{\theta}} = P_G^{-1}\left(H^{\otimes 2n}\right)^{-1}\boldsymbol{\theta} = P_G^{-1} 2^{-2n} H^{\otimes 2n}\boldsymbol{\theta}$$

which can be efficiently solved by a classical algorithm in $O(N^2 \log N^2)$ using a fast Walsh–Hadamard transform.

- For matrix of size $N = 2^n$, need N^2 CNOT and R_y gates, respectively.
- One can apply R_y only for $\widehat{\boldsymbol{\theta}}$ above some threshold to approximately block encode the matrix
- Open question: the characterization of matrices which have highly compressible FABLE circuits?
  - ➢Generally, you need matrices which are sparse in the Walsh-Hadamard domain: matrices such that $H^{\otimes n} A H^{\otimes n}$ is sparse.

S-FABLE method to block encode a general matrix

- How to transform matrices into Walsh-Hadamard sparse matrices?
  - If A is a sparse matrix, then the matrix $H^{\otimes n}AH^{\otimes n}$ is sparse in the Walsh-Hadamard domain because $H^{\otimes n}(H^{\otimes n}AH^{\otimes n})H^{\otimes n} = \text{A}$ is sparse
- Block encode HAH and then sandwich it with $H(HAH)H$
- Claims from the S-FABLE paper:
  - S-FABLE can drastically compress the size of the FABLE oracle, potentially at an exponential advantage (for unstructured sparse matrix)
  - Not so great for sparse matrix contains symmetries or structure

S-FABLE and LS-FABLE: Fast approximate block-encoding algorithms for unstructured sparse matrices
https://arxiv.org/pdf/2401.04234

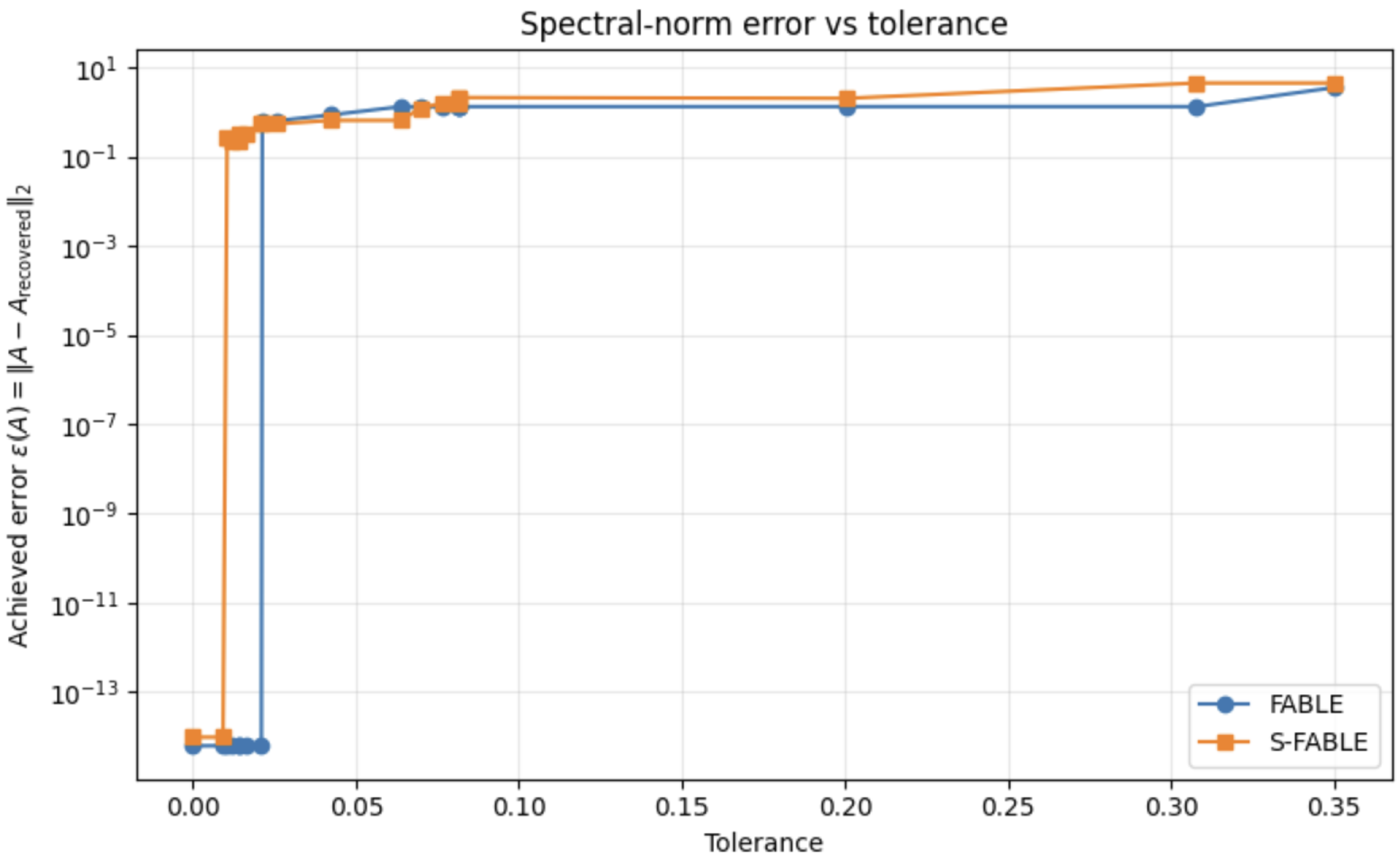
Testing of FABLE and S-FABLE

- Example 8x8 matrix A to encode

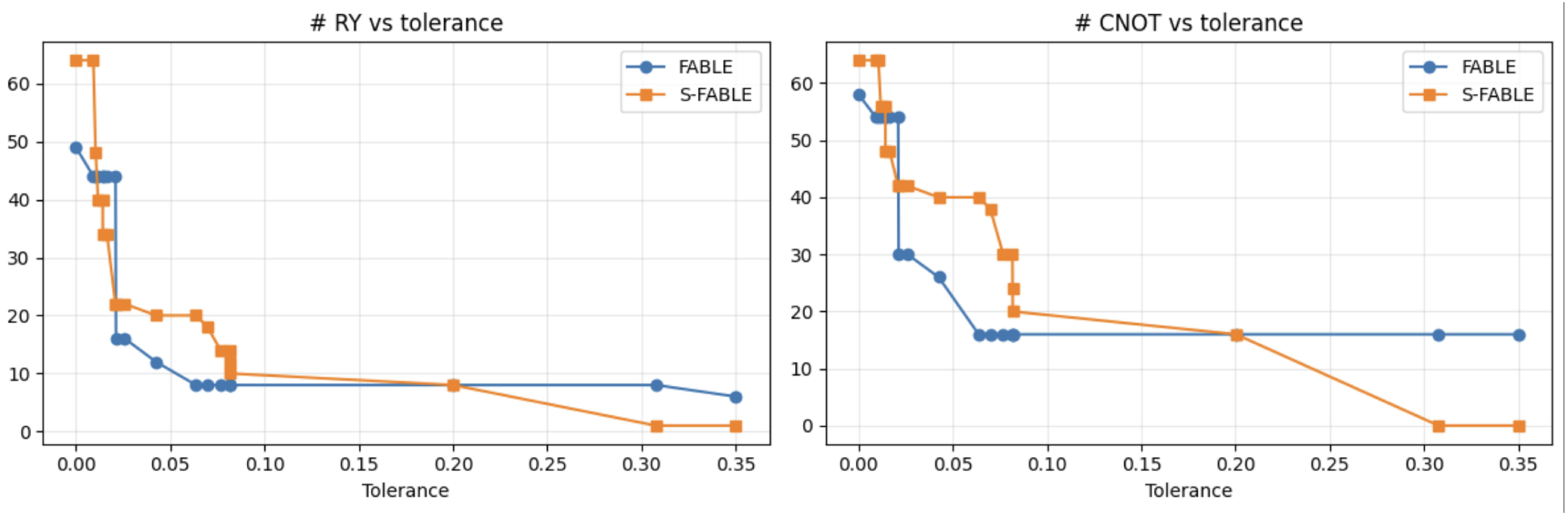
$$\begin{bmatrix} 3 & 0 & -1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 3 & 0 & 0 & -1 & 0 & 0 & 0 \\ -1 & 0 & 3 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 3 & 0 & 0 & -1 & 0 \\ 0 & -1 & 0 & 0 & 3 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 3 & 0 & -1 \\ 0 & 0 & 0 & -1 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 3 \end{bmatrix}$$

- Column and row qubit number = 3

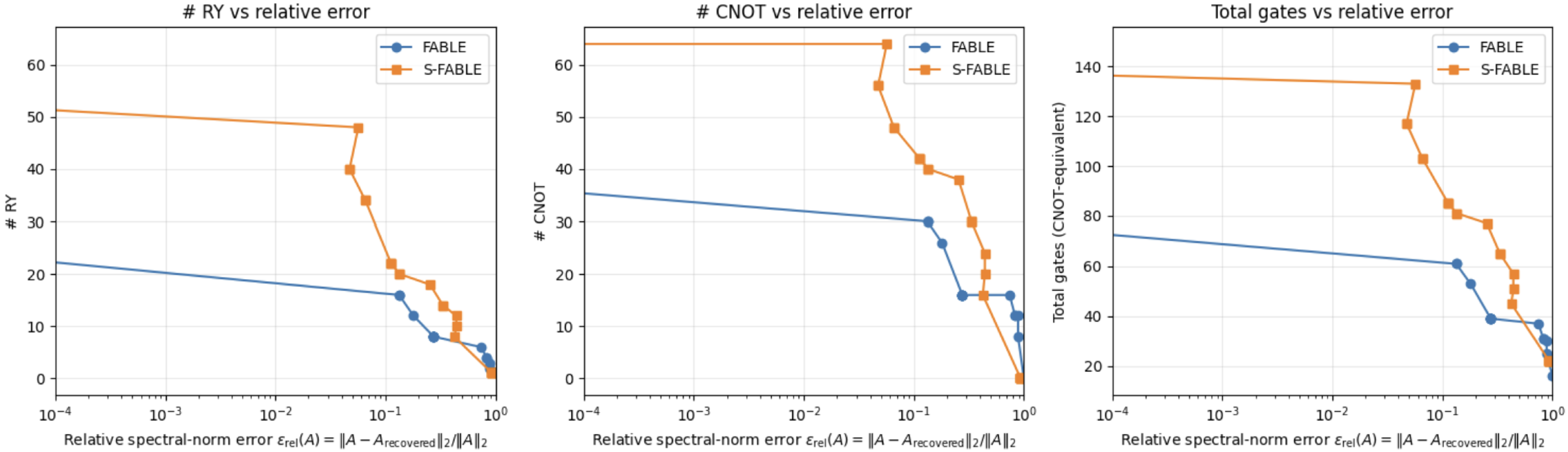
Testing of FABLE and S-FABLE



Testing of FABLE and S-FABLE



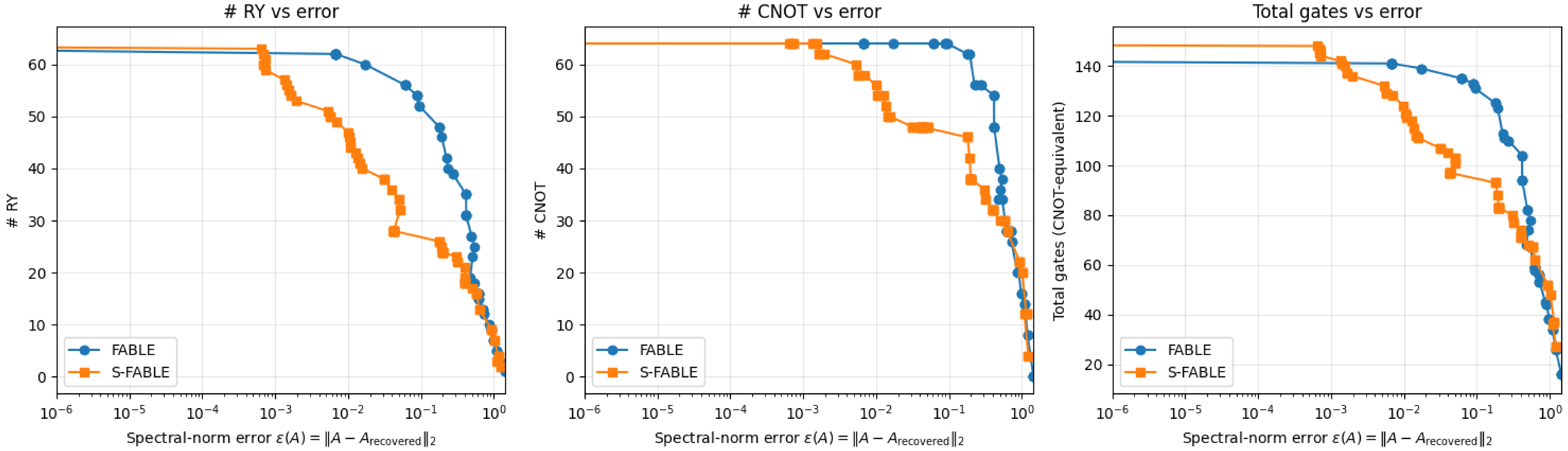
Testing of FABLE and S-FABLE



S-FABLE method performs worse than FABLE method for our matrix

Sanity check with a random matrix

$$
\begin{bmatrix}
0.72 & 0.00 & -0.31 & 0.00 & 0.00 & 0.18 & 0.00 & 0.00 \\
0.00 & -0.55 & 0.00 & 0.44 & 0.00 & 0.00 & -0.27 & 0.00 \\
-0.31 & 0.00 & 0.91 & 0.00 & -0.63 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.44 & 0.00 & -0.12 & 0.00 & 0.00 & 0.00 & 0.58 \\
0.00 & 0.00 & -0.63 & 0.00 & 0.37 & 0.29 & 0.00 & 0.00 \\
0.18 & 0.00 & 0.00 & 0.00 & 0.29 & -0.80 & 0.00 & 0.41 \\
0.00 & -0.27 & 0.00 & 0.00 & 0.00 & 0.00 & 0.66 & -0.35 \\
0.00 & 0.00 & 0.00 & 0.58 & 0.00 & 0.41 & -0.35 & -0.14
\end{bmatrix}
$$



Second method

Block encoding of sparse matrix

- Example 8x8 matrix A to encode

$$\begin{bmatrix} 3 & 0 & -1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 3 & 0 & 0 & -1 & 0 & 0 & 0 \\ -1 & 0 & 3 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 3 & 0 & 0 & -1 & 0 \\ 0 & -1 & 0 & 0 & 3 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 3 & 0 & -1 \\ 0 & 0 & 0 & -1 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 3 \end{bmatrix}$$

- Column qubit number = 3
- Max non-zero entries per column = 3, so slot qubit number = 2
- Scale the matrix to: $A' = \frac{A}{3}, A'_{jj} = 1, A'_{ij} = -\frac{1}{3}$ on edges

EXPLICIT QUANTUM CIRCUITS FOR BLOCK ENCODINGS OF CERTAIN SPARSE MATRICES

https://arxiv.org/pdf/2203.10236

Block encoding of sparse matrix

- We need the oracle function (slot model) that does:

$$c(j,\ell) = \begin{cases} j, & \ell = 0 \\ \text{neighbors}[j][\ell-1], & 1 \le \ell \le \deg(j) \\ \text{next smallest unused padding row}, & \text{otherwise (padding)} \end{cases}$$

$$\overbrace{\begin{bmatrix} 3 & 0 & -1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 3 & 0 & 0 & -1 & 0 & 0 & 0 \\ -1 & 0 & 3 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 3 & 0 & 0 & -1 & 0 \\ 0 & -1 & 0 & 0 & 3 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 3 & 0 & -1 \\ 0 & 0 & 0 & -1 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 3 \end{bmatrix}}^{j} \Big\} i$$

- For the example matrix,

	$\ell = 0$ (diagonal)	$\ell = 1$	$\ell = 2$	$\ell = 3$
$c(j=0,\ell)$	0	2	5	1
$c(j=1,\ell)$	1	4	0	2
$c(j=2,\ell)$	2	0	3	1
$c(j=3,\ell)$	3	2	6	0
$c(j=4,\ell)$	4	1	0	2
$c(j=5,\ell)$	5	0	7	1
$c(j=6,\ell)$	6	3	0	1
$c(j=7,\ell)$	7	5	0	1

- O_C must act reversibly ⇒ four slot values must map to four **distinct** row values. There exist a unique ℓ such that $i = c(j,\ell)$ as well as a unique ℓ' such that $j = c(i,\ell')$.
  - ➢Based on the table: $c(j=0,\ell=2) = 5 = i$. Then, $c(i=5,\ell'=1) = 0 = j$

Block encoding of sparse matrix

- We need the oracle function (slot model) that does:

$$c(j,\ell) = \begin{cases} j, & \ell = 0 \\ \text{neighbors}[j][\ell-1], & 1 \le \ell \le \deg(j) \\ \text{next smallest unused padding row}, & \text{otherwise (padding)} \end{cases}$$

$$\overbrace{\begin{bmatrix} 3 & 0 & -1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 3 & 0 & 0 & -1 & 0 & 0 & 0 \\ -1 & 0 & 3 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 3 & 0 & 0 & -1 & 0 \\ 0 & -1 & 0 & 0 & 3 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 3 & 0 & -1 \\ 0 & 0 & 0 & -1 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 3 \end{bmatrix}}^{j} \left.\vphantom{\begin{bmatrix} 0\\0\\0\\0\\0\\0\\0\\0 \end{bmatrix}}\right\} i$$

- For the example matrix,

	$\ell = 0$ (diagonal)	$\ell = 1$	$\ell = 2$	$\ell = 3$
$c(j=0,\ell)$	0	2	5	1
$c(j=1,\ell)$	1	4	0	2
$c(j=2,\ell)$	2	0	3	1
$c(j=3,\ell)$	3	2	6	0
$c(j=4,\ell)$	4	1	0	2
$c(j=5,\ell)$	5	0	7	1
$c(j=6,\ell)$	6	3	0	1
$c(j=7,\ell)$	7	5	0	1

- O_C must act reversibly ⇒ four slot values must map to four **distinct** row values. There exist a unique ℓ such that $i = c(j,\ell)$ as well as a unique ℓ' such that $j = c(i,\ell')$.
  - ➢Based on the table: $c(j=0,\ell=2) = 5 = i$. Then, $c(i=5,\ell'=1) = 0 = j$

Block encoding of general sparse matrix

- A is $2^n \times 2^n$, Hermitian, and s-sparse, with $s = 2^m$ (always padded to a power of two)
- We have an oracle O_C that, given a column index j and a slot index $\ell \in \{0, \ldots, s-1\}$, it returns the row index of the ℓ-th nonzero in column j:

$$O_C|\ell\rangle|j\rangle = |c(j,\ell)\rangle|j\rangle = |i\rangle|j\rangle$$

- There is an oracle O_A that, given (ℓ, j) (or equivalently (i, j)), performs a controlled rotation on the ancilla qubit:

$$O_A|0\rangle_a|\ell\rangle|j\rangle = \left(A_{c(j,\ell),j}|0\rangle_a + \cdots |1\rangle_a\right)|\ell\rangle|j\rangle$$

- Registers:
  - ➢Ancillas: two qubits. q_0 "spectator" ancilla (used in the Hermitian construction), q_1 "amplitude" ancilla that gets rotated.
  - ➢Slot/work register: n qubits, but only the last m (W_{lo}) are used for ℓ and the inactive part is denoted as W_{hi}.
  - ➢System register: n-qubit holding column j
- Initial state:

$$|0\rangle_a|0^n\rangle_W \otimes |\Psi\rangle_J$$

where $|\Psi\rangle = \sum_{j=0}^{2^n-1} \psi_j |\text{j}\rangle$.

Block encoding of general sparse matrix

- D_s applies Hadamards to the m slot qubits inside the n-qubit work register

$$|\Psi_0\rangle = |0\rangle_a|0^n\rangle \otimes \sum_j \psi_j|\mathrm{j}\rangle = \sum_j \psi_j|0\rangle_a|0^n\rangle_W|\mathrm{j}\rangle_J \xrightarrow{D_s} |\Psi_1\rangle = \frac{1}{\sqrt{s}}\sum_j\sum_{\ell=0}^{s-1}\psi_j|0\rangle_a|\mathbf{0^{n-m}}\rangle_{W_{hi}}|\ell\rangle_{W_{lo}}|j\rangle_J$$

- O_A rotates the amplitude ancilla a conditioned on (ℓ, j) so that:

$$|\Psi_1\rangle \xrightarrow{O_A} |\Psi_2\rangle = \frac{1}{\sqrt{s}}\sum_j\sum_{\ell=0}^{s-1}\psi_j\left(A_{c(j,\ell),j}|0\rangle_a + \cdots |1\rangle_a\right)|\mathbf{0^{n-m}}\rangle_{W_{hi}}|\ell\rangle_{W_{lo}}|j\rangle_J$$

- O_C compute the row index

$$|\Psi_2\rangle \xrightarrow{O_C} |\Psi_3\rangle = \frac{1}{\sqrt{s}}\sum_j\sum_{\ell=0}^{s-1}\psi_j\left(A_{c(j,\ell),j}|0\rangle_a + \cdots |1\rangle_a\right)|\boldsymbol{c(j,\ell)}\rangle_W|j\rangle_J$$

- SWAP the two n-qubit registers and does not touch ancilla qubit

$$|\Psi_3\rangle \xrightarrow{\text{SWAP}} |\Psi_4\rangle = \frac{1}{\sqrt{s}}\sum_j\sum_{\ell=0}^{s-1}\psi_j\left(A_{c(j,\ell),j}|0\rangle_a + \cdots |1\rangle_a\right)|\boldsymbol{j}\rangle_W|\boldsymbol{c(j,\ell)}\rangle_J = \cdots |\boldsymbol{j}\rangle_W|\boldsymbol{i}\rangle_J$$

Block encoding of general sparse matrix

- Apply $O_C^\dagger$ to uncompute. Since $O_C|\ell'\rangle|i\rangle = |c(i,\ell')\rangle|i\rangle = |j\rangle|i\rangle \Rightarrow O_C^\dagger|j\rangle|i\rangle = |\ell'\rangle|i\rangle$

$$|\Psi_4\rangle \xrightarrow{O_C^\dagger} |\Psi_5\rangle = \frac{1}{\sqrt{s}}\sum_j \sum_{\ell=0}^{s-1} \psi_j\big(A_{c(j,\ell),j}|0\rangle_a + \cdots |1\rangle_a\big)|\mathbf{0^{n-m}}\rangle_{\boldsymbol{W_{hi}}}|\ell'\rangle_{\boldsymbol{W_{lo}}}|\boldsymbol{i}\rangle_{\boldsymbol{J}}$$

- Apply $D_s^\dagger$, $D_s^\dagger|\ell'\rangle = \frac{1}{\sqrt{s}}\sum_{t=0}^{s-1}(-1)^{t\cdot\ell'}|t\rangle$, where $t \cdot \ell'$ is the bitwise inner product mod 2

$$|\Psi_5\rangle \xrightarrow{D_s^\dagger} |\Psi_6\rangle = \frac{1}{s}\sum_j \sum_{\ell=0}^{s-1}\sum_{t=0}^{s-1} \psi_j(-1)^{t\cdot\ell'}\big(A_{i,j}|0\rangle_a + \cdots |1\rangle_a\big)|\mathbf{0^{n-m}}\rangle_{\boldsymbol{W_{hi}}}|\boldsymbol{t}\rangle_{\boldsymbol{W_{lo}}}|\boldsymbol{i}\rangle_{\boldsymbol{J}}$$

- Inner product $\langle 0|_a\langle 0^n|_W \otimes I_J|\Psi_6\rangle = \langle 0|_a\langle 0^n|_{W_{\text{hi}}}\langle 0^m|_{W_{\text{lo}}} \otimes I_J|\Psi_6\rangle$

$$\langle 0|_a\langle 0^n|_W \otimes I_J|\Psi_7\rangle = \frac{1}{s}\sum_j\sum_{\ell=0}^{s-1}\psi_j A_{i,j}|i\rangle_J = \frac{1}{s}\sum_j\sum_{\ell=0}^{s-1}\psi_j A_{c(j,\ell),j}|\boldsymbol{c}(\boldsymbol{j},\boldsymbol{\ell})\rangle_{\boldsymbol{J}}$$

- The ℓ-sum: for a fixed column j, ℓ enumerates the (padded) nonzero rows i. So this is exactly:

$$\frac{1}{s}\sum_j\sum_{\ell=0}^{s-1}\psi_j A_{c(j,\ell),j}|c(j,\ell)\rangle_J = \frac{1}{s}\sum_j\sum_i \psi_j A_{i,j}|i\rangle_J = \frac{1}{s}\sum_i\Big(\sum_j A_{i,j}\,\psi_j\Big)|i\rangle_J = \frac{A}{s}|\Psi\rangle_J$$

Block encoding of general sparse matrix

- O_A can be done easily with gray code
- O_C is very difficult to implement

$$O_C = \sum_j |j\rangle\langle j| \otimes P_j$$

- For column 0, we need complex permutation gate if there is no structure:

  $000 \mapsto 000, 001 \mapsto 010, 010 \mapsto 101, 011 \mapsto 001, 100 \mapsto 011, 101 \mapsto 100, 110 \mapsto 110, 111 \mapsto 111.$
- Each column j needs quantum multiplexer

Thank you