VQA_POISSON:

Efficient and Scalable Quantum Library Solving Two-Dimensional Poisson Equations with Mixed Boundary Conditions

Dongyun Chung, Jiyong Choi, Jung-Il Choi

2025 / 08 / 12

School of Mathematics and Computing (Computational Science and Engineering)

YONSEI UNIVERSITY

Contents

Introduction

- VQA_POISSON provides:

Unified Framework

- A unified framework conducting optimization with Variational Quantum Algorithms using `QuantumComputer`, `QuantumCalculator`, `QuantumOptimizer` classes.
- Can be further scaled with the same structure for other optimization problems, assuming that the corresponding cost function and quantum circuits are designed.
- Supports execution on both IBM hardware & IonQ Simulator

Integrated Library

- Our library integrates algorithms from several references[1], [2], [3] solving Poisson equations, as well as subroutines[4] that optimize circuits for better compatibility with IBM hardware architecture
- Adapts the quantum circuits according to boundary conditions.
- Provides stability analysis for the configuration provided.

Test runs & Examples

- A Jupyter Notebook & test run output for simple 1D cases on IonQ simulator

(1) Yuki Sato, Ruho Kondo, Satoshi Koide, Hideki Takamatsu, and Nobuyuki Imoto. Variational quantum algorithm based on the minimum potential energy for solving the Poisson equation. *Physical Review A,* 104(5):052409, 2021. doi: 10.1103/PhysRevA.104.052409.
(2) Minjin Choi and Hoon Ryu. A variational quantum algorithm for tackling multi-dimensional Poisson equations with inhomogeneous boundary conditions. *New J. Phys.*, 2025, 27, 054510.
(3) Xiaoqi Liu, Yuedi Qu, Ming Li, Shen Ming and Shu-Qian Shen. A variational quantum algorithm for the Poisson equation based on the banded Toeplitz systems. *Commun. Theor. Phys.*, 77(4):045101, 2025. doi: 10.1088/1572-9494/ad8bae
(4) Byeongyong Park and Doyeol Ahn. Reducing CNOT count in quantum Fourier transform for the linear nearest-neighbor architecture. *Sci Rep* **13**, 8638 (2023). https://doi.org/10.1038/s41598-023-35625-3

Method Summary

Poisson Equation

$$\nabla^2 \psi(x_1, \ldots, x_n) = f(x_1, \ldots, x_n)$$

- The Poisson equation is a multi-dimensional PDE where f is the known source function and p is the unknown scalar field.

Cost Function

- The cost function uses the energy functional from FEM methods and central difference scheme from FDM methods:

$$E_{\text{discrete}}(\boldsymbol{\theta}) = -\frac{1}{2}\frac{\left[\langle f, \psi(\boldsymbol{\theta}) | X \otimes I^{\otimes n} | f, \psi(\boldsymbol{\theta}) \rangle\right]^2}{\langle \psi(\boldsymbol{\theta}) | \boldsymbol{A} | \psi(\boldsymbol{\theta}) \rangle}$$

- For mathematical detail, refer to Sato et al. (2021)(1)

Boundary Conditions

- For general Robin boundary conditions, we adopt the method from Choi et al. (2025)(2)
- For periodic boundary conditions, we adopt the QFT method from Liu et al. (2025)(3) with the LNN-optimized QFT from Park et al. (2023)(4)

(1) Yuki Sato, Ruho Kondo, Satoshi Koide, Hideki Takamatsu, and Nobuyuki Imoto. Variational quantum algorithm based on the minimum potential energy for solving the Poisson equation. *Physical Review A,* 104(5):052409, 2021. doi: 10.1103/PhysRevA.104.052409.
(2) Minjin Choi and Hoon Ryu. A variational quantum algorithm for tackling multi-dimensional Poisson equations with inhomogeneous boundary conditions. *New J. Phys.*, 2025, 27, 054510.
(3) Xiaoqi Liu, Yuedi Qu, Ming Li, Shen Ming and Shu-Qian Shen. A variational quantum algorithm for the Poisson equation based on the banded Toeplitz systems. *Commun. Theor. Phys.*, 77(4):045101, 2025. doi: 10.1088/1572-9494/ad8bae
(4) Byeongyong Park and Doyeol Ahn. Reducing CNOT count in quantum Fourier transform for the linear nearest-neighbor architecture. *Sci Rep* **13**, 8638 (2023). https://doi.org/10.1038/s41598-023-35625-3

Installation & Execution

Requirements

- Make sure your Python environment matches the following criteria:
  - ➔ `Python` >= 3.8
  - ➔ `qiskit` 1.4.3
  - ➔ `qiskit-ionq` 0.5.13
  - ➔ `qiskit-ibm-runtime` 0.39.0
  - ➔ `NumPy, SciPy, Matplotlib`

❖ Important note:

- `qiskit`'s source code, library functions, and internal structure can change significantly even between minor releases. Using a newer (or older) version than the ones listed above may lead to incompatibilities or runtime errors.

- For IonQ simulators, our main code automatically retrieves the backend using the user's `IONQ_API_KEY` from the operating system environment variables
- For IBM hardware execution, the backend is automatically chosen as the least busy available device via the `QiskitRuntimeService().least_busy()` method.

- For more details, refer to the official documentation of IBM and IonQ.

Installation & Execution

Run Instructions

- Clone our git (https://github.com/MPMC-Lab/VQA_POISSON.git) via git clone

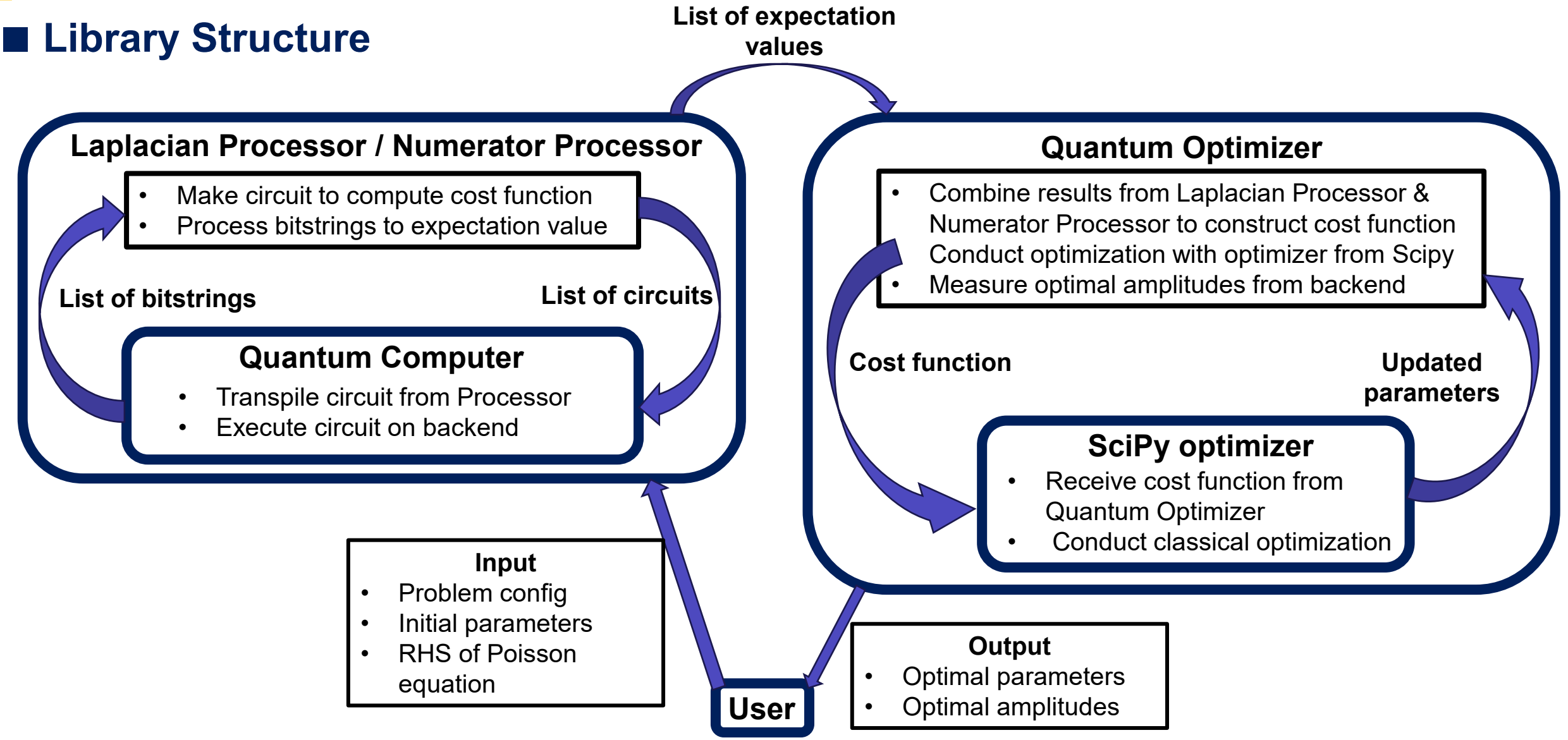
- To execute the main script, move inside to VQA_POISSON via `cd VQA_POISSON` and run `main_1D.py` or `main_2D.py`.

- The user can modify problem settings (grid size, ansatz depth, backend, etc) via `input_1D` or `input_2D`.

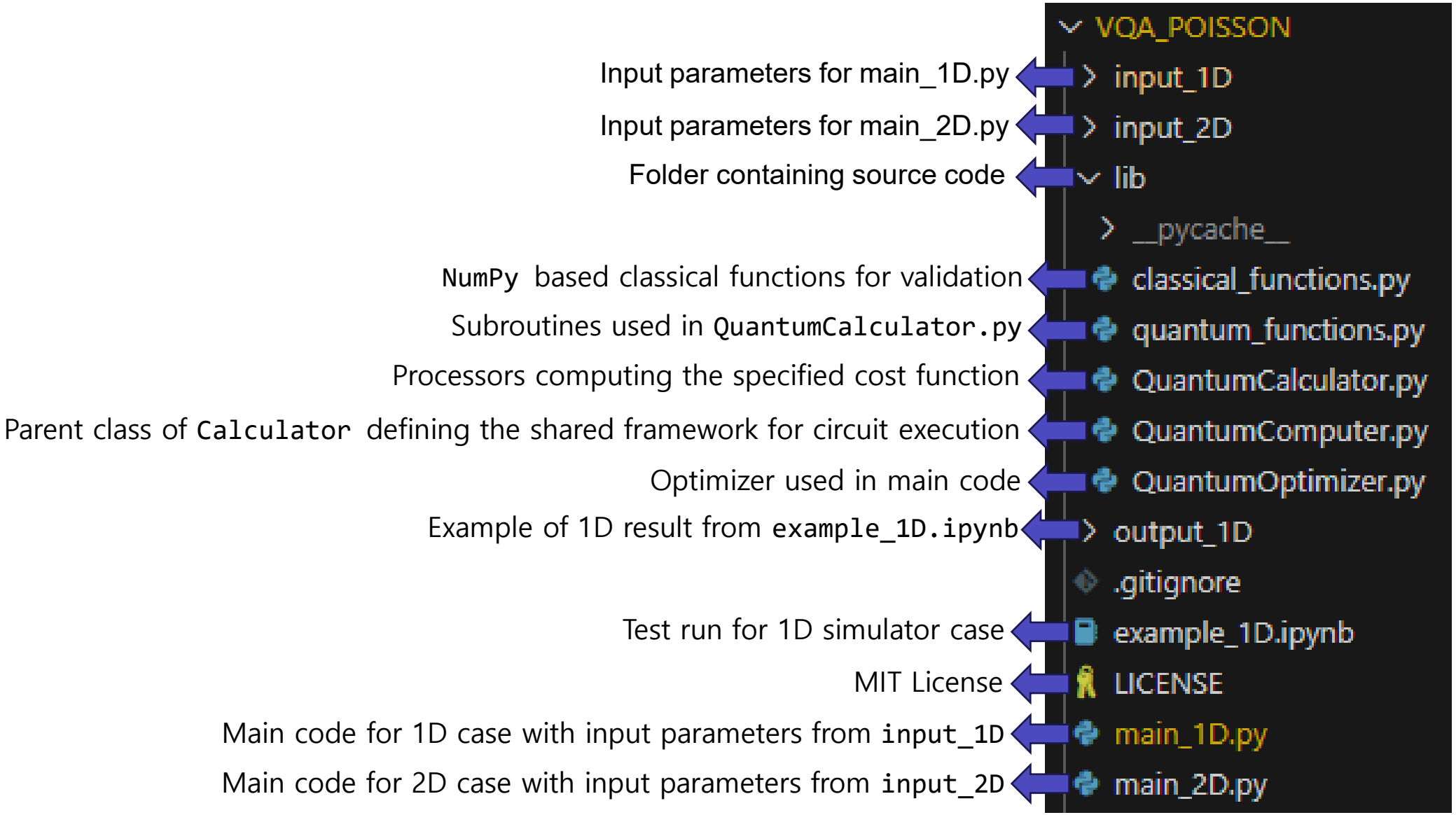
  ➔ For details about configurations, refer to Page 9.

Overview

Library Structure



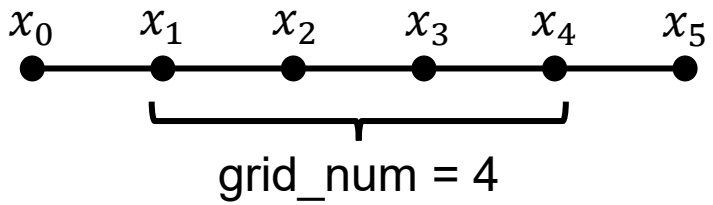
Library Structure



Configuration

input_1D / config.yaml

- `grid_num`
  - ➔ Number of grid points **excluding** the boundary nodes (Make sure this is a power of 2):



- `ansatz_depth`
  - ➔ Depth of the default LNN ansatz. For user-defined ansatz, this parameter is ignored.
- `boundary_condition`
  - ➔ "R" for Robin boundary conditions, "P" for periodic boundary conditions.
- `method`
  - ➔ Type of the classical optimizer in `SciPy.optimize.minimize`.
- `x0, x1`
  - ➔ Coordinates of the edges in 1D domain.

```yaml
# Number of interior grid points (excluding boundary, must be a power of 2)
grid_num: 16

# Depth of the variational ansatz
ansatz_depth: 4

# Type of boundary condition: 'P' or 'R'
boundary_condition: R

# Type of optimizer
method: Powell

# Length of 1D Domain
x0: 0.0
x1: 1.0

# Boundary constants (Only used for 'R')
alpha: 0.0
beta: 0.0
gamma: 0.0

# Number of shots per circuit execution
num_shots: 524288

# Backend type: either 'simulator' or 'hardware'
backend: hardware
```

Configuration

input_1D / config.yaml

- alpha, beta, gamma
  - ➔ Constants only used for Robin boundary conditions. Make sure that

$$\alpha u(x_0) + \beta \frac{\partial u(x_0)}{\partial x} = \gamma,$$

$$\alpha u(x_n) - \beta \frac{\partial u(x_N)}{\partial x} = \gamma$$

  where x_0, x_N are the boundary nodes.

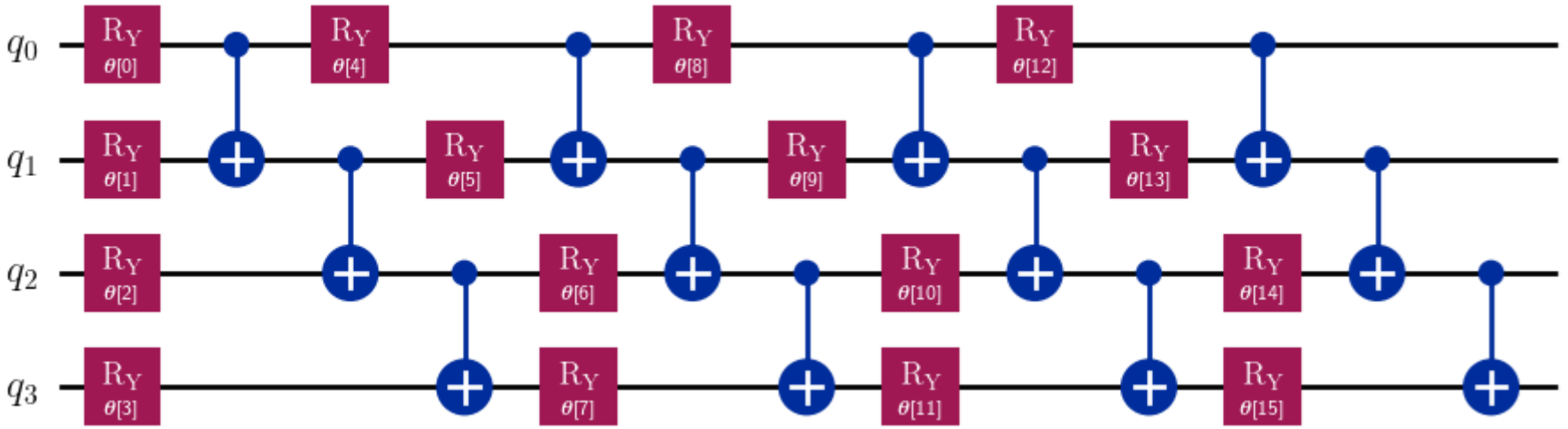
- num_shots
  - ➔ Number of shots applied for each quantum circuit
- backend
  - ➔ IBM hardware or IonQ Simulator

```yaml
# Number of interior grid points (excluding boundary, must be a power of 2)
grid_num: 16

# Depth of the variational ansatz
ansatz_depth: 4

# Type of boundary condition: 'P' or 'R'
boundary_condition: R

# Type of optimizer
method: Powell

# Length of 1D Domain
x0: 0.0
x1: 1.0

# Boundary constants (Only used for 'R')
alpha: 0.0
beta: 0.0
gamma: 0.0

# Number of shots per circuit execution
num_shots: 524288

# Backend type: either 'simulator' or 'hardware'
backend: hardware
```

Configuration

input_1D / initial_params.csv

⟨Default LNN design⟩



input_1D > initial_params.csv > data

1	5.169140526378443568e+00
2	1.022415443229351517e+01
3	5.288208587760161450e+00
4	1.088438925971671578e+00
5	1.014437814122117310e+00
6	3.970149737721972993e+00
7	6.875673017518111330e+00
8	5.963358260905656572e+00
9	4.061654249831773811e+00
10	1.253188667237613574e+00
11	2.851522898973699593e+00
12	6.944346516102055666e+00
13	7.266916952370207072e-01
14	1.203378485529325737e+01
15	8.403990813831674345e+00
16	8.403429176482553942e+00

- `initial_params.csv` specifies the initial parameters for VQA optimization.
- Make sure that the length of `initial_params.csv` matches the number of parameters in the ansatz.
- For the default design, $\text{param\_num} = \log_2(\text{grid\_num}) \times \text{ansatz\_depth}$

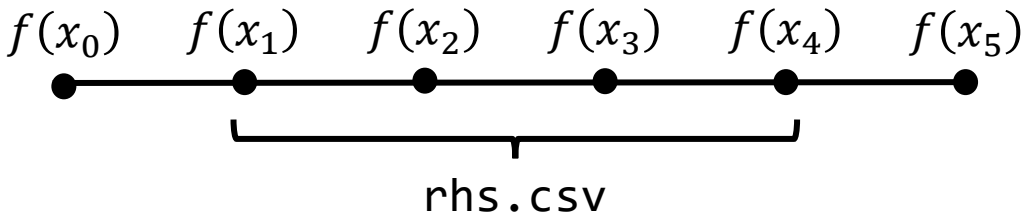
Configuration

input_1D / rhs.csv

- The source function $f(\boldsymbol{x})$ of the Poisson equation:

$$\nabla^2\psi(\boldsymbol{x}) = f(\boldsymbol{x})$$

- The data specified in `rhs.csv` corresponds to $f(\boldsymbol{x})$ for each node:

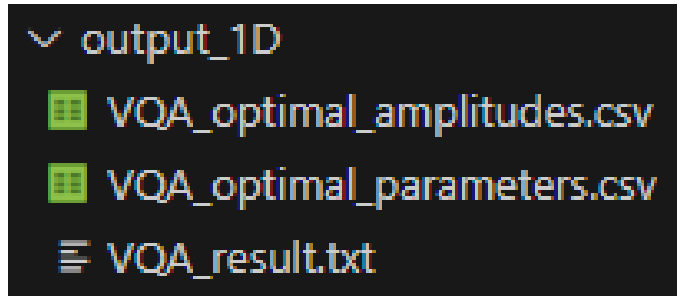


- Make sure the length of `rhs.csv` is equal to `grid_num` in `config.yaml`
- `rhs.csv` will be automatically normalized in the main code to be implemented as a `StatePreparation` block from `qiskit`

input_1D > rhs.csv > data

```
1   -1.8135350497404700 84e+00
2   -3.5653123384575771 13e+00
3   -5.1956771916093398 29e+00
4   -6.6491094895287892 19e+00
5   -7.8761143385101783 92e+00
6   -8.8349075600515583 06e+00
7   -9.4928386000390575 14e+00
8   -9.8275024086765583 32e+00
9   -9.8275024086765601 08e+00
10  -9.4928386000390575 14e+00
11  -8.8349075600515583 06e+00
12  -7.8761143385101792 80e+00
13  -6.6491094895287901 07e+00
14  -5.1956771916093433 82e+00
15  -3.5653123384575811 09e+00
16  -1.8135350497404703 06e+00
```

Output_1D

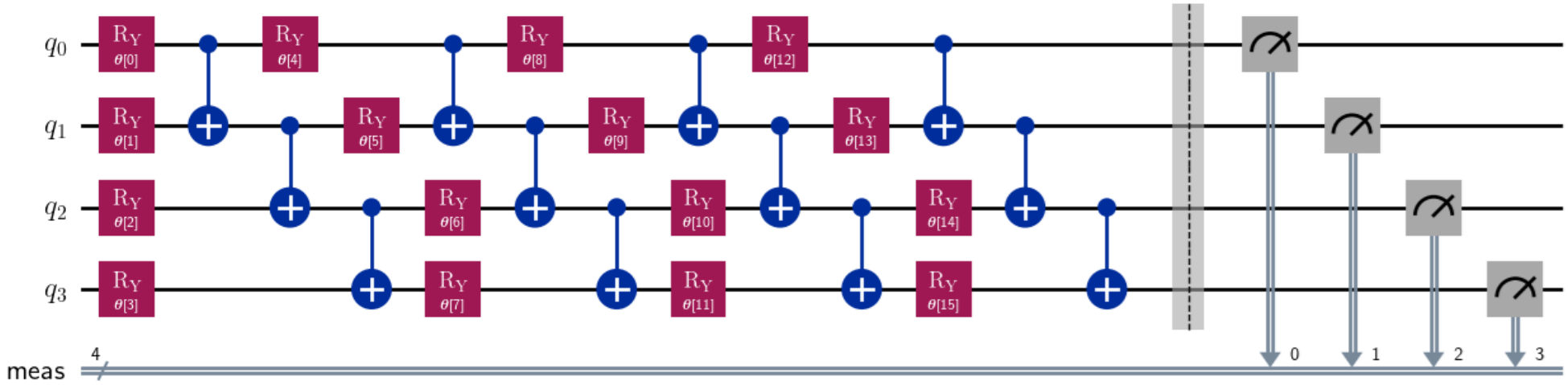


VQA_optimal_parameters.csv

- The optimal parameters after VQA convergence.

VQA_optimal_amplitudes.csv

- Reads out the amplitude of each computational basis state after applying the optimal parameters to the predefined ansatz:



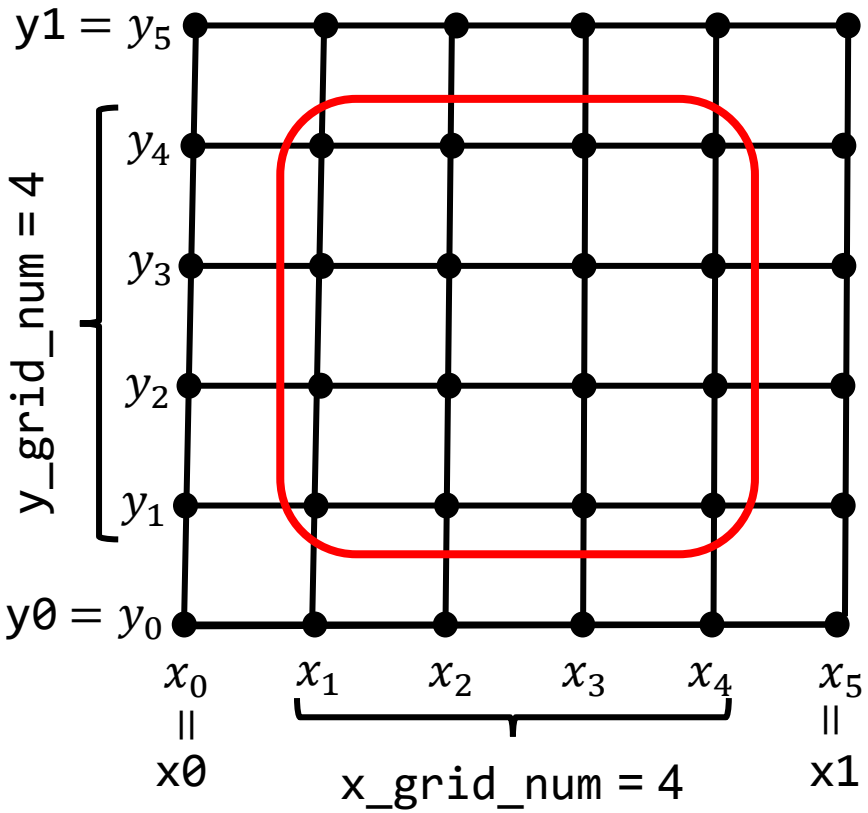
- If the solution is positive throughout the domain, then the optimal amplitudes correspond to the VQA solution. Otherwise, sign corrections are required.
- Note that the amplitudes follow `qiskit`'s convention, in which the last qubit is the most significant.

  ➔ Ex) $|q_0 q_1 q_2 q_3\rangle = |0001\rangle = |8\rangle$, not $|1\rangle$.
- `VQA_result.txt` contains the result message from `SciPy.optimize.minimize`

Configuration

input_2D / config.yaml

- x_grid_num / y_grid_num
  - ➔ Number of grid points along the x (or y) direction **excluding** the boundary nodes (Make sure this is a power of 2):




```yaml
# Number of grid points along the x & y direction (excluding boundary, must be a power of 2)
x_grid_num: 32
y_grid_num: 16

# Depth of the variational ansatz
ansatz_depth: 4

# Type of boundary condition: A two-letter combination of 'P' or 'R'
boundary_condition: PR

# Type of optimizer
method: Powell

# x-directional length of domain
x0: 0.0
x1: 1.0

# y-directional width of domain
y0: 0.0
y1: 1.0

# Boundary constants (Only used for 'R')
alpha_x: 0.0
alpha_y: 0.0
beta_x: 0.0
beta_y: 0.0
gamma_x: 0.0
gamma_y: 0.0

# Number of shots per circuit execution
num_shots: 524288

# Backend type: either 'simulator' or 'hardware'
backend: simulator
```

Configuration

input_2D / config.yaml

- ansatz_depth
  - ➔ Depth of the default LNN ansatz. For user-defined ansatz, this parameter is ignored.
- boundary_condition
  - ➔ Ex) "PR" : Periodic along x, Robin along y
    "RP" : Robin along x, Periodic along y
- method
  - ➔ Type of the classical optimizer in `SciPy.optimize.minimize`.
- alpha, beta, gamma
  - ➔ Constants only used for Robin boundary conditions. Make sure that

$$\alpha_x u(x_0) + \beta_x \frac{\partial u(x_0)}{\partial x} = \gamma_x,$$

$$\alpha_x u(x_n) - \beta_x \frac{\partial u(x_N)}{\partial x} = \gamma_x,$$

$$\alpha_y u(y_0) + \beta_y \frac{\partial u(y_0)}{\partial y} = \gamma_y,$$

$$\alpha_y u(y_N) - \beta_y \frac{\partial u(y_N)}{\partial y} = \gamma_y$$

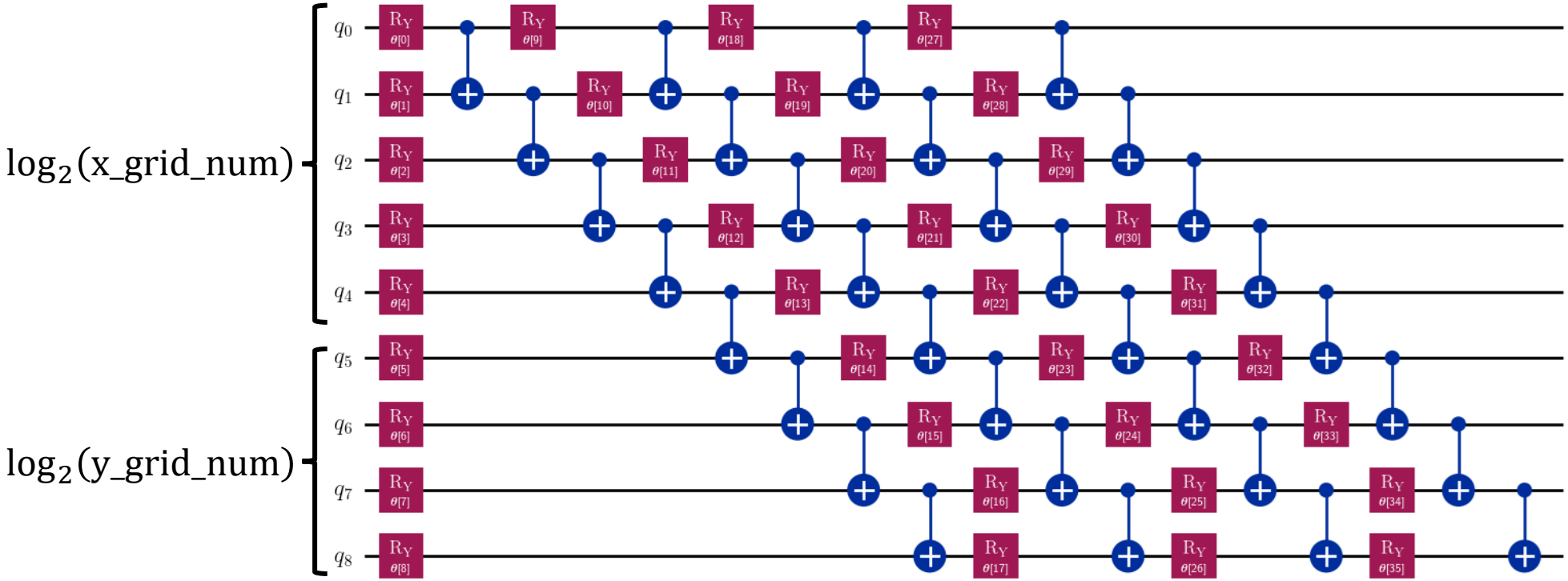
```yaml
# Number of grid points along the x & y direction (excluding boundary, must be a power of 2)
x_grid_num: 32
y_grid_num: 16

# Depth of the variational ansatz
ansatz_depth: 4

# Type of boundary condition: A two-letter combination of 'P' or 'R'
boundary_condition: PR

# Type of optimizer
method: Powell

# x-directional length of domain
x0: 0.0
x1: 1.0

# y-directional width of domain
y0: 0.0
y1: 1.0

# Boundary constants (Only used for 'R')
alpha_x: 0.0
alpha_y: 0.0
beta_x: 0.0
beta_y: 0.0
gamma_x: 0.0
gamma_y: 0.0

# Number of shots per circuit execution
num_shots: 524288

# Backend type: either 'simulator' or 'hardware'
backend: simulator
```

Configuration

input_2D / initial_params.csv

- The first $\log_2(\text{x\_grid\_num})$ qubits correspond to x-directional operations, where the last $\log_2(\text{y\_grid\_num})$ qubits correspond to y-directional operations.
- Make sure that the length of `initial_params.csv` matches the number of parameters in the ansatz.
- For the default design, $\text{param\_num} = \log_2(\text{x\_grid\_num} \times \text{y\_grid\_num}) \times \text{ansatz\_depth}$

⟨**Default LNN design**⟩



Configuration

input_2D / rhs.csv

- The source function $f(x, y)$ of the Poisson equation:

$$\nabla^2 \psi(x, y) = f(x, y)$$

- The data specified in `rhs.csv` corresponds to $f(x, y)$ for each interior node displayed in **Page 14**
- The data must be ordered so that:
  - ➔ Increasing the row index corresponds to increasing y (from top to bottom)
  - ➔ Increasing the column index corresponds to increasing x (from left to right)

$f(x_1, y_1)$	$f(x_2, y_1)$	$\cdots$	$f(x_{N-1}, y_1)$
$f(x_1, y_2)$	$f(x_2, y_2)$	$\cdots$	$f(x_{N-1}, y_2)$
$\vdots$	$\vdots$	$\ddots$	$\vdots$
$f(x_1, y_{N-1})$	$f(x_2, y_{N-1})$	$\cdots$	$f(x_{N-1}, y_{N-1})$

y_grid_num

x_grid_num

- Make sure the dimension of `rhs.csv` matches (`x_grid_num , y_grid_num`) in `config.yaml`
- `rhs.csv` will be automatically normalized in the main code to be implemented as a `StatePreparation` block from `qiskit`

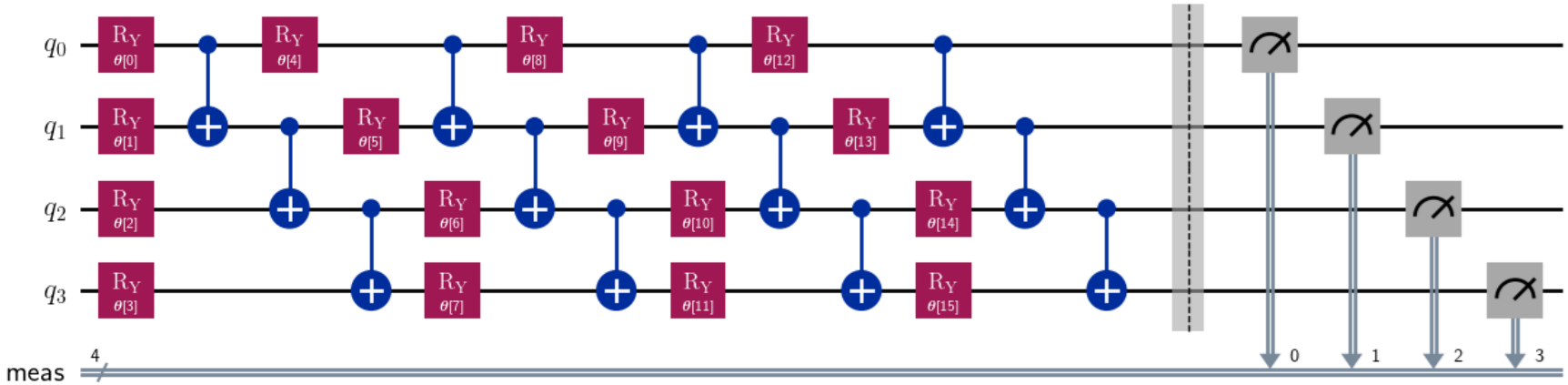
Output_2D

VQA_optimal_parameters.csv

- The optimal parameters after VQA convergence.

VQA_optimal_amplitudes.csv

- Reads out the amplitude of each computational basis state after applying the optimal parameters to the predefined ansatz:



- If the solution is positive throughout the domain, then the optimal amplitudes correspond to the VQA solution. Otherwise, sign corrections are required.

  ➔ **Note that in 2D, you may need to reshape the amplitude array to reconstruct the 2D domain.**

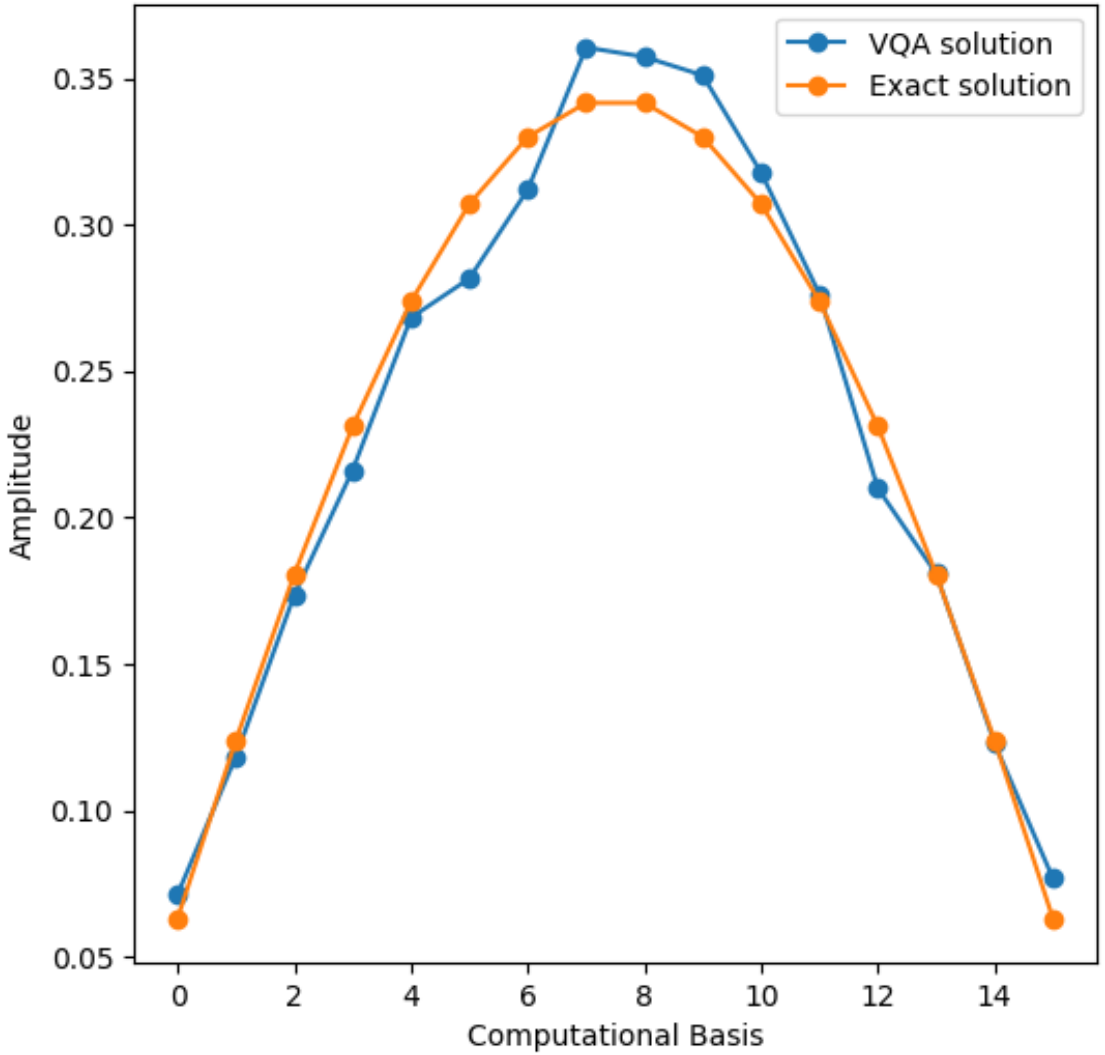
- Note that the amplitudes follow `qiskit`'s convention, in which the last qubit is the most significant.

  ➔ Ex) $|q_0 q_1 q_2 q_3\rangle = |0001\rangle = |8\rangle$, not $|1\rangle$.

- `VQA_result.txt` contains the result message from `SciPy.optimize.minimize`

Validation

1D Validation (IonQ Simulator)



- Solution to $\nabla^2\psi(x) = f(x)$ where $f(x) = -\pi^2\sin(\pi x)$ with homogeneous Dirichlet boundary conditions.
- For details, refer to `example_1D.ipynb`

License & Contact

License

Contact

- Lead Developer : **Dongyun Chung** (achung3312@yonsei.ac.kr)
- Corresponding Author: **Prof. Jung-Il Choi** (jic@yonsei.ac.kr)
- Department: Computational Science Engineering, Yonsei University, Seoul, 03722, South Korea

Q&A

Thanks for listening

Appendix

User-defined Ansatz

- To use other ansatz designs, you may design a parametrized `QuantumCircuit` object from `qiksit` and replace the `psi_param_circuit` variable with that circuit.
- Then the processors will automatically take those ansatz, retrieve the parameters, and apply operations to compute the cost function.
- However, make sure that the number of qubits is correctly designed
  - ➔ For 1D, $\text{num\_qubits} = \log_2(\text{grid\_num})$
  - ➔ For 2D, $\text{num\_qubits} = \log_2(\text{x\_grid\_num} \times \text{y\_grid\_num})$

```
parameters = ParameterVector(r'$\boldsymbol{\theta}$', length=param_num)
psi_param_circuit = make_LNN_ansatz(num_qubits, ansatz_depth, parameters)
```

➔ Replace this line with your ansatz

```
laplacian_processor = LaplacianEVProcessor1D(
                        ansatz_list=[psi_param_circuit],
                        boundary_condition_list=[boundary_condition],
                        dx_list = [dx],
                        backend=used_backend,
                        num_shots=num_shots,
                        is_simulator = is_simulator,
                        sampler = sampler,
                        alpha = alpha,
                        beta = beta
                    )
```

➔ Then the processors with refer to that ansatz