QUANTUM COMPUTING INTERNSHIP EXPERIENCE

August 2023 - December 2023

National Nuclear Laboratory (NNL)

NNL Mini Apps
1st Milestone Update

Mandy Bowman
Yury Chernyak
Horia Mărgărit

Key Results and Progress

Progress

- Conducted literature review
- Narrowed focus to specific problem
- Selected a paper with accompanying GitHub repository
- Defined project such that we will be able to provide results (positive or negative) within the next two months

Evolution of the Poisson Equation in Quantum Computing

2009 HHL linear solver algorithm is presented - requires fault-tolerance

2012 Cao - uses HHL in solving the Poisson Eqn

2019 Wang - uses HHL in solving the Poisson Eqn

2020 Lubasch - Variational Quantum Algorithms for nonlinear problems

2020 Lui - VQA to solve Poisson Eqn

2022 Sato - VQA to solve Poisson Eqn

Chosen Papers

- Y. Cao et al. Quantum algorithm and circuit design solving the Poisson equation, New J. Phys. 15 013021. (2013)
- Sato, Y., Kondo, R., Koide, S., Takamatsu, H., & Imoto, N. Variational quantum algorithm based on the minimum potential energy for solving the Poisson equation. Physical Review A. (2021)

What did we learn?

VQA Poisson ~ Intro to Problem

Consider the poisson equation such that x is defined on the d-dimensional cubic domain:

$$-\nabla^2 u(\boldsymbol{x}) = f(\boldsymbol{x}) \quad \text{in} \quad \Omega.$$

$x \in I_d = (0,1)^d$ ← *d-dimensional cube*

Del operator is the second spatial derivative hence— the definition being:

$$f'(x) \approx \frac{f(x+\delta x) - f(x)}{\delta x}$$

$$f''(x) \approx \frac{f(x+\delta x) - 2f(x) + f(x-\delta x)}{\delta x^2}$$

i.e. take two neighboring points and subtract one from the other and divide by the grid spacing

Represented as a vectors:

$$\frac{1}{\delta x^2}\begin{pmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_{i+1}) \\ \vdots \\ f(x_N) \\ f(x_0) \end{pmatrix} - \frac{2}{\delta x^2}\begin{pmatrix} f(x_0) \\ f(x_1) \\ \vdots \\ f(x_i) \\ \vdots \\ f(x_{N-1}) \\ f(x_N) \end{pmatrix} + \frac{1}{\delta x^2}\begin{pmatrix} f(x_N) \\ f(x_0) \\ \vdots \\ f(x_{i-1}) \\ \vdots \\ f(x_{N-2}) \\ f(x_{N-1}) \end{pmatrix}$$

VQA Poisson: Discretization

Transforming to quantum

Amplitude encoding function & Shift/adder operator:

$$P := \sum_{i \in [0, 2^n - 1]} |(i+1) \bmod 2^n\rangle \langle i|$$

The second order derivative finite difference in the quantum version is equivalent to:

$$\hat{\Delta} |\psi\rangle = \frac{1}{\delta x^2} \left(\hat{A} |\psi\rangle - 2 |\psi\rangle + \hat{A}^\dagger |\psi\rangle \right)$$

Note: These A's are supposed to be P's – shift operators

Now the Poisson equation is now equivalent to:

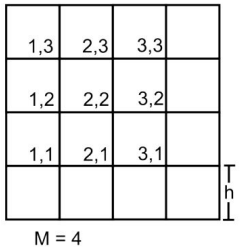
$$\lambda_0 \hat{\Delta} |\psi\rangle = |f\rangle$$

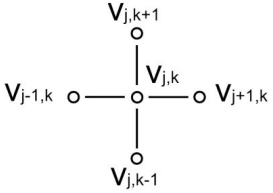
In Liu et al., they minimize the distance between the two vectors, and plugging in the quantum laplace operator:

Minimize $\left|\left| |f\rangle - \lambda_0 \left(\hat{A} |\psi\rangle - 2 |\psi\rangle + \hat{A}^\dagger |\psi\rangle \right) \right|\right|^2$

Quantum algorithm and circuit design solving the Poisson equation: Yudong Cao

Discretization:





$$f'(x) \approx \frac{f(x+\delta x) - f(x)}{\delta x}$$

$$f''(x) \approx \frac{f(x+\delta x) - 2f(x) + f(x-\delta x)}{\delta x^2}$$

$$\frac{1}{\delta x^2}\begin{pmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_{i+1}) \\ \vdots \\ f(x_N) \\ f(x_0) \end{pmatrix} - \frac{2}{\delta x^2}\begin{pmatrix} f(x_0) \\ f(x_1) \\ \vdots \\ f(x_i) \\ \vdots \\ f(x_{N-1}) \\ f(x_N) \end{pmatrix} + \frac{1}{\delta x^2}\begin{pmatrix} f(x_N) \\ f(x_0) \\ \vdots \\ f(x_{i-1}) \\ \vdots \\ f(x_{N-2}) \\ f(x_{N-1}) \end{pmatrix}$$

$$h^{-2}\left((-v_{j-1,k} + 2v_{j,k} - v_{j+1,k}) + (-v_{j,k-1} + 2v_{j,k} - v_{j,k+1})\right) = f_{j,k}$$

The Poisson equation w/ Dirichlet boundary condition can be recast as a linear system. I.e $\boldsymbol{Au} = \boldsymbol{f}$

I.e Above is equivalent to: $-\Delta_h \vec{v} = \vec{f}_h$

M = 4 example:

$$h^{-2}A\begin{pmatrix} v_1 \\ \vdots \\ v_9 \end{pmatrix} := h^{-2}\begin{pmatrix} B & -I & \\ -I & B & -I \\ & -I & B \end{pmatrix}\begin{pmatrix} v_1 \\ \vdots \\ v_9 \end{pmatrix} = \begin{pmatrix} f_1 \\ \vdots \\ f_9 \end{pmatrix} \quad \Rightarrow \quad A = \begin{pmatrix} L_h + 2I & -I & 0 & \cdots & \cdots & 0 \\ -I & L_h + 2I & -I & 0 & \cdots & 0 \\ 0 & -I & \ddots & \ddots & 0 & \vdots \\ \vdots & 0 & \ddots & \ddots & -I & 0 \\ \vdots & \vdots & 0 & -I & L_h + 2I & -I \\ 0 & 0 & \cdots & 0 & -I & L_h + 2I \end{pmatrix}$$

$$-\Delta_h = h^{-2}A \quad \Leftarrow$$

$$A = L_h \otimes I + I \otimes L_h$$

This is the explanation of how the Sato paper shows the **A** matrices for *periodic, Neumann and Dirichlet*

Relating to *Sato* from previous slide:

$$
\begin{aligned}
A_{\text{periodic}} &= I^{\otimes n-1} \otimes (I - X) \\
&\quad + P^{-1} \left(I^{\otimes n-1} \otimes (I - X) \right) P \\
A_{\text{Dirichlet}} &= A_{\text{periodic}} + P^{-1} \left(I_0^{\otimes n-1} \otimes X \right) P \\
A_{\text{Neumann}} &= A_{\text{periodic}} - P^{-1} \left(I_0^{\otimes n-1} \otimes (I - X) \right) P.
\end{aligned}
$$

- Liu showed that the matrix A-Dirichlet can be decomposed into O(n)
- Here, decomposition of A-periodic, A-Dirichlet, and A-Neumann into O(1) terms

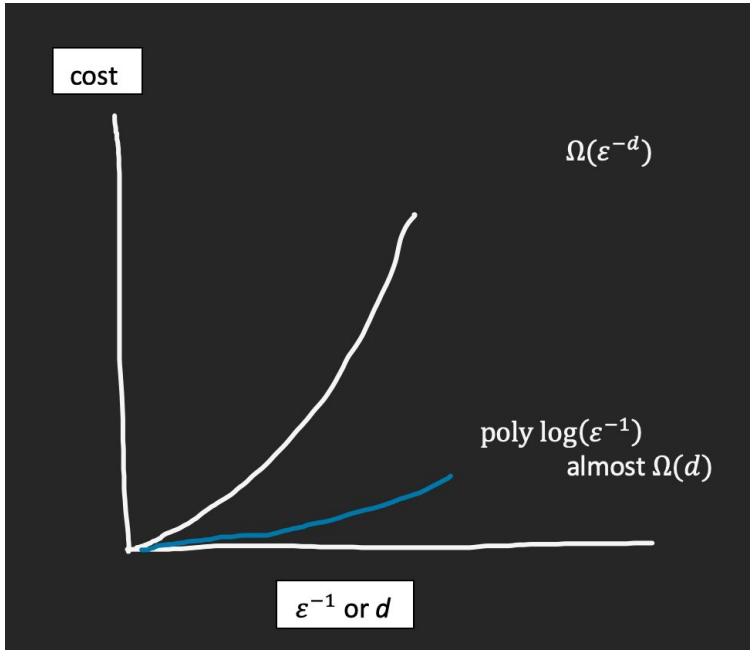
$$
A_{\text{Dirichlet}} := \begin{bmatrix}
2 & -1 & 0 & & \dots & & 0 \\
-1 & 2 & -1 & 0 & \dots & & 0 \\
0 & -1 & 2 & -1 & \dots & & 0 \\
\vdots & & & \ddots & & & \vdots \\
0 & & & \dots & 0 & -1 & 2 & -1 \\
0 & & & \dots & & 0 & -1 & 2
\end{bmatrix}
$$

$$
A_{\text{periodic}} := \begin{bmatrix}
2 & -1 & 0 & & \dots & & 0 & -1 \\
-1 & 2 & -1 & 0 & \dots & & & 0 \\
0 & -1 & 2 & -1 & \dots & & & 0 \\
\vdots & & & \ddots & & & & \vdots \\
0 & & & \dots & 0 & -1 & 2 & -1 \\
-1 & 0 & & \dots & & 0 & -1 & 2
\end{bmatrix}
$$

$$
A_{\text{Neumann}} := \begin{bmatrix}
1 & -1 & 0 & & \dots & & 0 \\
-1 & 2 & -1 & 0 & \dots & & 0 \\
0 & -1 & 2 & -1 & \dots & & 0 \\
\vdots & & & \ddots & & & \vdots \\
0 & & & \dots & 0 & -1 & 2 & -1 \\
0 & & & \dots & & 0 & -1 & 1
\end{bmatrix}
$$

Quantum versus Classical

- Classically to solve *d* dimensional problem with error epsilon will be at least exponentially
  - ➢ matrix A grows exponentially as d increases.
- Quantum algorithm: finds a solution that is **polynomial in the logarithm of inverse error** and ~ **linear** in dimension *d*
- achieving an **exponential** speed up over classical algorithms.



https://arxiv.org/pdf/1207.2485.pdf

Sato Paper: Problem

- Improves on the previous papers (Liu, Yao)
- As before, paper solves the Poisson equation
  - Discretizing the equation => i.e. derives a system matrix
  - Decomposing the matrix and mapping the components into quantum states and into a quantum circuit

- Introduces the ***Energy Minimization Method:***
- Cost function is based on the minimum potential energy of a system

- Reasoning behind this is derived $\Rightarrow$

$$-\nabla^2 u(\boldsymbol{x}) = f(\boldsymbol{x}) \quad \text{in} \quad \Omega.$$

Total P.E Equation:

$$E := \frac{1}{2}\int_\Omega \nabla v^* \cdot \nabla v \, d\Omega - \frac{1}{2}\int_\Omega v^* f \, d\Omega - \frac{1}{2}\int_\Omega f^* v \, d\Omega,$$

Minimizing means derivative = 0:

$$\begin{aligned}
0 =& dE(v;\delta v) \\
=& \frac{1}{2}\int_\Omega \nabla \delta v^* \cdot \nabla v \, d\Omega + \frac{1}{2}\int_\Omega \nabla v^* \cdot \nabla \delta v \, d\Omega \\
& - \frac{1}{2}\int_\Omega \delta v^* f \, d\Omega - \frac{1}{2}\int_\Omega f^* \delta v \, d\Omega \\
=& \frac{1}{2}\int_{\Gamma_N} \delta v^* \boldsymbol{n} \cdot \nabla v \, d\Gamma - \frac{1}{2}\int_\Omega \delta v^* \left(\nabla^2 v + f\right) \, d\Omega \\
& + \frac{1}{2}\int_{\Gamma_N} \delta v \left(\boldsymbol{n} \cdot \nabla v\right)^* \, d\Gamma - \frac{1}{2}\int_\Omega \delta v \left(\nabla^2 v + f\right)^* \, d\Omega
\end{aligned}$$

Applying Dirichlet and Neumann— you get that the first and third terms become equal to zero and hence vanish

$$-\boldsymbol{n} \cdot \nabla u(\boldsymbol{x}) = 0 \quad \text{on} \quad \Gamma_N,$$
$$u(\boldsymbol{x}) = 0 \quad \text{on} \quad \Gamma_D,$$

Conclusion: minimizing the total potential energy w.r.t function v yields the state field u which is governed by poisson's equation.

How to get to Quantum-version

After Discretization:

$$E_h := \frac{1}{2}\boldsymbol{v}^* \cdot A\boldsymbol{v} - \frac{1}{2}\boldsymbol{v}^* \cdot \boldsymbol{f} - \frac{1}{2}\boldsymbol{f}^* \cdot \boldsymbol{v},$$

- v and f denote vectors with component values of v and f at the nodes discretizing the domain Ω

After

- ➢ encoding f and v into quantum states with parameterized solution state
- ➢ Preparing f as $|f, \psi(\theta)\rangle := \sqrt{}(|0\rangle |f\rangle + |1\rangle |\psi(\theta)\rangle) / 2$
- ➢ applying the necessary condition for optimality (requiring partial of $Eh(r,\theta)$ w.r.t r is equal to zero)

You get the final form of **Cost function:**

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

⇐ The **A** matrix is the important part, gives rise to the three boundary conditions discussed.

Compared to Cost function in Liu: (does not provide norm to the solution)

$$E(\boldsymbol{\theta}) = \langle \psi(\boldsymbol{\theta}) \,|\, A\,(I - |f\rangle\,\langle f|)\,A \,|\, \psi(\boldsymbol{\theta})\rangle$$

VQA

The Proposed Algorithm Outline:

Step 1 Initialize a set of parameters θ in a classical computer.

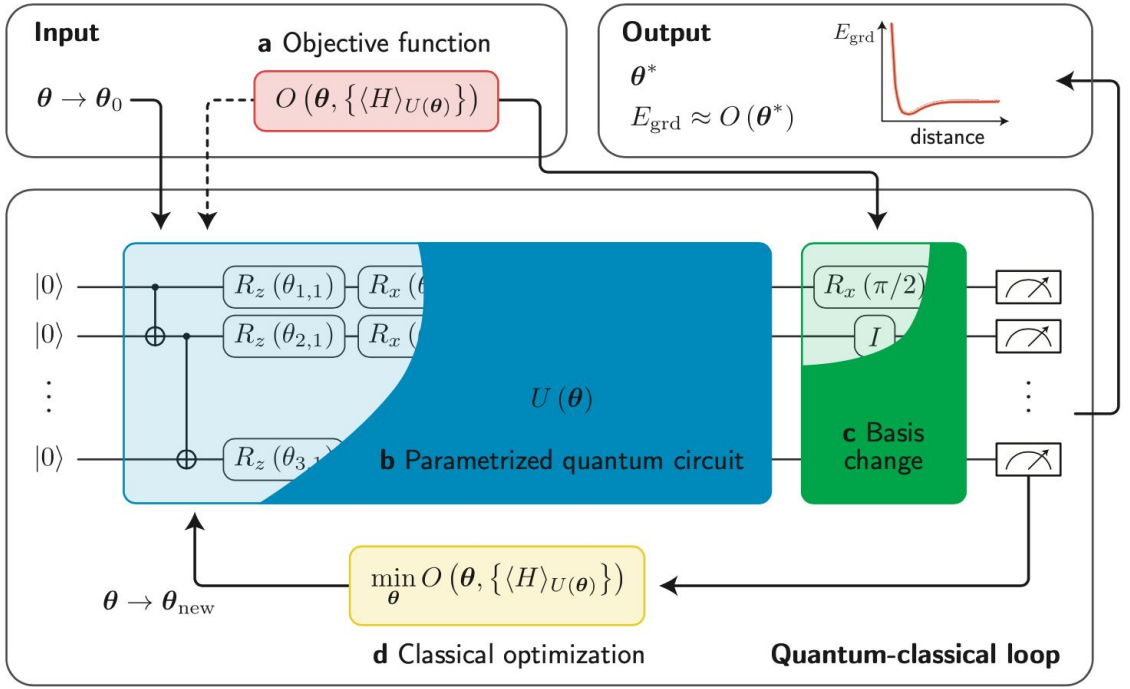
Step 2 Evaluate the cost function E_h using a quantum computer.

Step 3 If a certain terminal condition is satisfied, the optimization procedure halts; otherwise, proceed to Step 4.

Step 4 Update the set of parameters using some classical optimization scheme, then return to Step 2.

Terminal conditions:

- Optimization procedure was terminated when the norm of the gradients became less than the predetermined threshold value
- The optimization was performed 10x from randomly set initial parameters θ between [0,4*pi] for each boundary condition



(Noisy intermediate-scale quantum algorithms Kishor Bharti et. al., 2022)

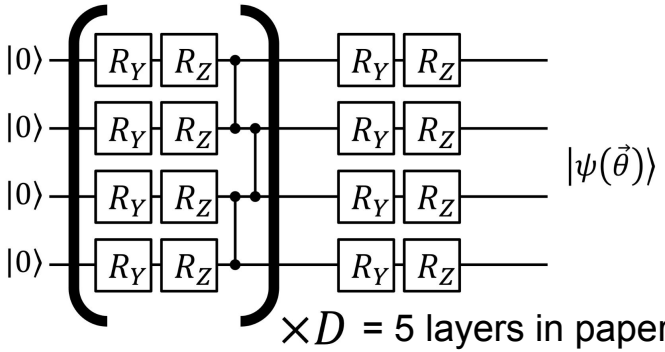
VQA workflow:

a) the **objective function** (*O*) — encodes the problem
b) the **parameterized quantum circuit** (*U*) — variables theta are tuned to minimize the objective function
c) the **measurement scheme** — performs basis changes & measurements needed to compute expectation values (used to evaluate the objective)
d) the **classical optimizer** — minimizes the objective.

Circuits

Hardware-Efficient Ansatz (HEA):

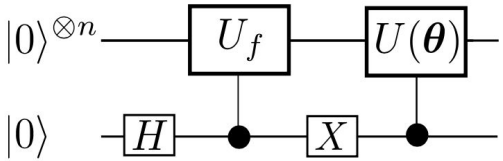
- suffers from barren plateaus at long depths, it can also avoid them at shallow ones
- one of the most NISQ- friendly ansatzes
- “shallow HEA should likely be avoided in VQA implementations if one seeks to find a quantum advantage” (**On the practical usefulness of the Hardware Efficient Ansatz** Lorenzo Leone, Salvatore F.E. Oliviero, Lukasz Cincio, M. Cerezo, 2022)



Circuit of the preparation of
|f, ψ(θ)⟩ := √ (|0⟩ |f ⟩ + |1⟩ |ψ(θ)⟩) / 2 :

Unitary that prepares the state:

$$U_{\mathrm{b}} = H^{\otimes n} X \otimes I^{\otimes (n-1)}$$

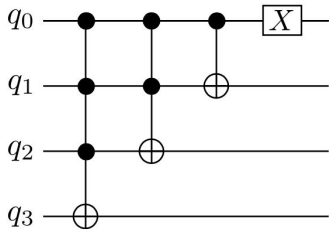


$$\hat{A} = \sum_i |(i+1) \mod 2^n\rangle\langle i|$$

$$\hat{A}^\dagger = \sum_i |(i-1) \mod 2^n\rangle\langle i|$$

$$\hat{A}|\psi\rangle = \hat{A}\begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_i \\ \vdots \\ x_{N-1} \end{pmatrix} = \begin{pmatrix} x_N \\ x_0 \\ \vdots \\ x_{i-1} \\ \vdots \\ x_{N-2} \end{pmatrix} \qquad \hat{A}^\dagger|\psi\rangle = \hat{A}^\dagger\begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_i \\ \vdots \\ x_{N-1} \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{i+1} \\ \vdots \\ x_0 \end{pmatrix}$$

Shift Operator Circuit:



VQA Poisson: Resources

$$f'(x) \approx \frac{f(x+\delta x) - f(x)}{\delta x}$$

$$f''(x) \approx \frac{f(x+\delta x) - 2f(x) + f(x-\delta x)}{\delta x^2}$$

Recall
- ➢ Finite Difference Method
- ➢ Requires Function Evaluation At Three Points
- ➢ For Each Gradient Estimate At The Central Point

$$h^{-2}\left((-v_{j-1,k} + 2v_{j,k} - v_{j+1,k}) + (-v_{j,k-1} + 2v_{j,k} - v_{j,k+1})\right) = f_{j,k}$$

Recall
- ➢ Periodic Boundary Conditions
- ➢ Requires That Function Values Are Equal At Periodic Intervals
- ➢ For Sufficiently Long Intervals Can Model Infinite Function Domains

$$-\Delta_h = h^{-2}A$$

$$A_{\text{periodic}} := \begin{bmatrix} 2 & -1 & 0 & \dots & & 0 & -1 \\ -1 & 2 & -1 & 0 & \dots & & 0 \\ 0 & -1 & 2 & -1 & \dots & & 0 \\ \vdots & & & \ddots & & & \vdots \\ 0 & & & \dots & 0 & -1 & 2 & -1 \\ -1 & 0 & & \dots & & 0 & -1 & 2 \end{bmatrix} \in \mathbb{R}^{N \times N}$$

$$E_h := \frac{1}{2}\boldsymbol{v}^* \cdot A\boldsymbol{v} - \frac{1}{2}\boldsymbol{v}^* \cdot \boldsymbol{f} - \frac{1}{2}\boldsymbol{f}^* \cdot \boldsymbol{v},$$

A_{periodic} may be the discretization matrix for E_h

$$E_h = \frac{1}{2}r^2 \langle \psi(\boldsymbol{\theta}) \,|\, A \,|\, \psi(\boldsymbol{\theta}) \rangle - \frac{1}{2}r \langle \psi(\boldsymbol{\theta}) \,|\, f \rangle - \frac{1}{2}r \langle f \,|\, \psi(\boldsymbol{\theta}) \rangle$$

$$E_h(r, \boldsymbol{\theta}) = \frac{1}{2}r^2 \langle \psi(\boldsymbol{\theta}) \,|\, A \,|\, \psi(\boldsymbol{\theta}) \rangle - r \langle f, \psi(\boldsymbol{\theta}) \,|\, X \otimes I^{\otimes n} \,|\, f, \psi(\boldsymbol{\theta}) \rangle$$

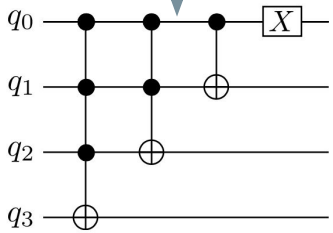
O(n) **qubits** are required to estimate the gradient at $O(2^n)$ discretization nodes! *Think: tensor product of measured n qubits.*

VQA Poisson: Resources (circuit depth)

$$P := \sum_{i \in [0, 2^n - 1]} |(i+1) \bmod 2^n\rangle \langle i|$$

$$A_{\text{periodic}} := \begin{bmatrix} 2 & -1 & 0 & & \dots & & 0 & -1 \\ -1 & 2 & -1 & 0 & \dots & & & 0 \\ 0 & -1 & 2 & -1 & \dots & & & 0 \\ \vdots & & & \ddots & & & & \vdots \\ 0 & & & \dots & 0 & -1 & 2 & -1 \\ -1 & 0 & & \dots & & 0 & -1 & 2 \end{bmatrix} = \begin{bmatrix} 1 & -1 & 0 & 0 & \dots & & 0 \\ -1 & 1 & 0 & 0 & \dots & & 0 \\ 0 & 0 & 1 & -1 & \dots & & 0 \\ 0 & 0 & -1 & 1 & \dots & & 0 \\ \vdots & & & \ddots & & & \vdots \\ 0 & & & \dots & 0 & 0 & 1 & -1 \\ 0 & & & \dots & 0 & 0 & -1 & 1 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 & & \dots & & 0 & -1 \\ 0 & 1 & -1 & 0 & \dots & & & 0 \\ 0 & -1 & 1 & 0 & \dots & & & 0 \\ \vdots & & & \ddots & & & & \vdots \\ 0 & & & & 0 & 1 & -1 & 0 \\ 0 & & & \dots & 0 & -1 & 1 & 0 \\ -1 & 0 & & \dots & & 0 & 0 & 1 \end{bmatrix}$$

$$A_{\text{periodic}} = A_{\mathcal{T}_{\text{even}}} + A_{\mathcal{T}_{\text{odd}}} \qquad A_{\mathcal{T}_{\text{even}}} = I^{\otimes n-1} \otimes (I - X) \qquad A_{\mathcal{T}_{\text{odd}}} = P^{-1} \left(I^{\otimes n-1} \otimes (I - X) \right) P$$



O(d) **gates** are required to estimate the gradient in D spatial dimensions!

$$f'(x) \approx \frac{f(x+\delta x) - f(x)}{\delta x}$$

$$f''(x) \approx \frac{f(x+\delta x) - 2f(x) + f(x-\delta x)}{\delta x^2}$$

$$A_d := A \otimes I^{\otimes(d-1)} + I \otimes A \otimes I^{\otimes(d-2)} + \cdots + I^{\otimes(d-1)} \otimes A$$

$$h^{-2} \left((-v_{j-1,k} + 2v_{j,k} - v_{j+1,k}) + (-v_{j,k-1} + 2v_{j,k} - v_{j,k+1}) \right) = f_{j,k}$$

VQA Poisson: Resources

- A run of a quantum circuit to obtain a sample is a shot
- Estimation of shots to evaluate expectation values:
  - The number of terms to be measured is O(d) for the boundary expectation values (i.e. the denominators in the cost function)
- The Cost function can be rewritten using the mean value (mu) and is assumed to be estimated as follows:

$$E_h = -\frac{1}{2}\frac{\mu_1^2}{\sum_{i=2}^{m}\mu_i} \approx -\frac{1}{2}\frac{\bar{q}_1^2}{\sum_{i=2}^{m}\bar{q}_i} =: g(\bar{q}_1, \ldots, \bar{q}_m)$$

  - m denotes boundary condition: m = 3 (periodic), m = 4 (Dirichlet), m = 5 (Neumann)
  - qi ^ (j) denotes the jth sample value for i-th expectation value. q1(j) is for the numerator expectation value in cost function equation.

- Mean square error is inversely proportional to the number of shots:

$$\varepsilon^2 \approx r_{\text{opt}}^2\left(\sigma_1^2 + \frac{1}{4}r_{\text{opt}}^2\sum_{i=2}^{m}\sigma_i^2\right)\frac{1}{S}$$

- Estimating cost function value with a quantum computer:
  - Number of Q-circuits required corresponds to the numerator and the number of terms in denominator (the A matrix eqn 24-26 in paper). Tc = 3, 4, 5 (periodic, Dirichlet, Neumann)

State Prep:
$$T_P := \mathcal{O}(D_{\text{ansatz}} + D_{\text{enc}} + D_{\text{shift}}) = \mathcal{O}(D_{\text{ansatz}} + D_{\text{enc}} + n^2)$$

Cost Function Eval:
Is constant.
$$T_C := \mathcal{O}(1)$$

Gradient Eval:
Scales linearly
$$T_G := \mathcal{O}(nD_{\text{ansatz}})$$

Num_shot:
$$T_S := \mathcal{O}\left(\frac{1}{\varepsilon^2}\right)$$

Total:
$$T := T_{\text{it}}T_P\left(T_C + T_G\right)T_S = \mathcal{O}\left(\frac{T_{\text{it}}\left(D_{\text{ansatz}} + D_{\text{enc}} + n^2\right)nD_{\text{ansatz}}}{\varepsilon^2}\right)$$

- Time complexity for solving the Poisson equation by classical computing is **O(N log N) ; *N* = *size of matrix = 2**n***
- Improves from classically and from Liu:

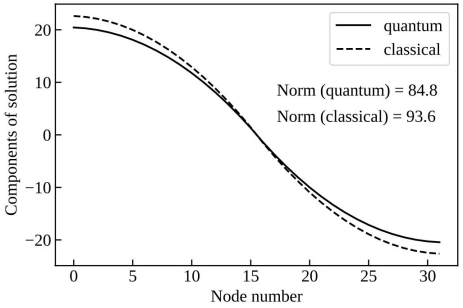
$$T = \mathcal{O}\left(\frac{T_{\text{it}}(D_{\text{ansatz}} + D_{\text{enc}})n^2 D_{\text{ansatz}}}{\varepsilon^2}\right)$$

Results of Sato Paper

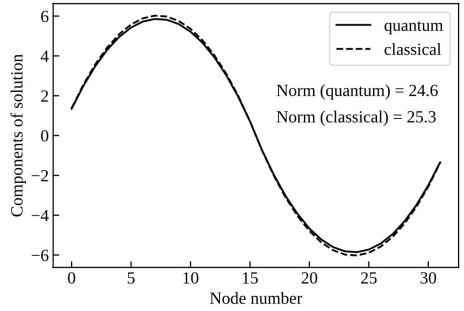
- The statevector simulator backend in Aer was used to evaluate the proposed method in an ideal environment without any noise or sampling errors.
- Optimization procedure was terminated when the norm of the gradients became < predetermined threshold value.
- The optimization was performed 10x from randomly set initial parameters θ for each boundary condition.
- Required number of quantum circuits: Tc = 3,4,5 (periodic, Dirichlet, Neumann) (independent of scale n)
- Num_parameters is O(nD_ansats), so num_of_Q-Circuits is proportional to this.
- Classical Optimizer used for updating parameters: Broyden–Fletcher–Goldfarb–Shanno (BFGS) method
    - The BFGS method is known for its efficiency and ability to converge to a local minimum in optimization problems.
    - The number of iterations is strongly dependent on the classical optimization solver and the terminal condition setting

Graphs show that the algorithm underestimates the norms of the solution vectors (although the directions of the solution vectors given by the proposed method are in good agreement with those from classical computing)

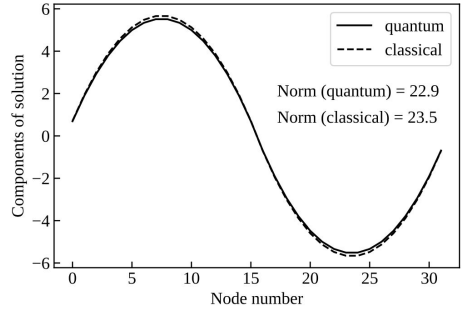
Classically solving Poisson eqn: O(NlogN) w/ N = size of matrix



(c) Neumann boundary conditions



(b) Dirichlet boundary conditions



(a) Periodic boundary conditions

Pt 2 results:

- The method significantly reduces the required number of expectation value calculations, and overall time complexity
- But, Neumann boundary condition gives a totally different solution, underestimates the norms of the solution vectors, although the directions of the solution vectors given by the proposed method are in good agreement with those from classical computing

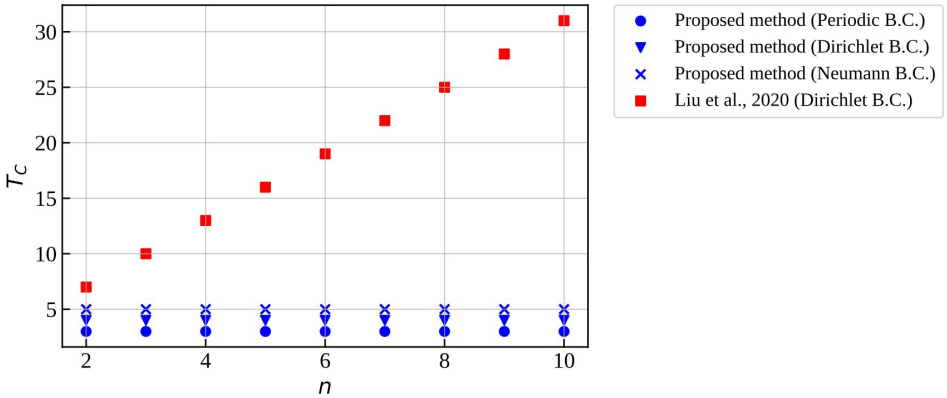


FIG. 5. Number of circuit executions per cost function evaluation T_C vs. the number of qubits for both the proposed and previous methods [26].

Final Conclusions of Paper:

I. provided decomposition of matrices into O(1) terms – hence smaller num_of_measurements needed
II. provided info about the norms of the solution vectors
III. Time complexity has improved

$$r = \frac{1}{\sqrt{\langle \psi(\boldsymbol{\theta}) \,|\, A^2 \,|\, \psi(\boldsymbol{\theta}) \rangle}}.$$

$$T_C := \mathcal{O}(1)$$

- Requires O(1) measurements per cost function evaluation–Liu had O(n) qubits

New direction, next steps, and challenges

New Direction

- Decided at last meeting with Brian to deep dive into Sato et al. paper

Next steps: Implementation on Quantinuum

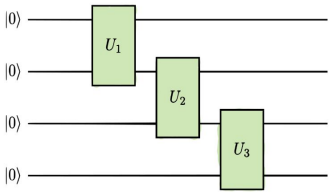
- Implementing VQA Poisson current code on Quantinuum (has not been done before)
  - Simulator (in progress)
  - Simulator with noise
  - quantum computer (have access Oct 16th-22nd)
- Challenges:
  - Need to update VQA Poisson (python and qiskit) to satisfy requirements of Quantinuum
  - Sato et al. noted a bottleneck of this algorithm is the gate depth of the shift operator

Next steps: Developing other Ansatz

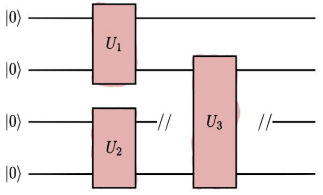
Developing another Ansatz to experiment with (likely Tensor network)

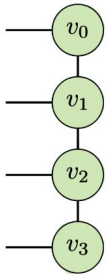
- Need to conduct literature review (in progress)
- Resources: PennyLane tensor network templates & Qiskit code (from Alberto)
- Challenges: Figure out how to translate the current 1D poisson function into tensor networks and represent it as such

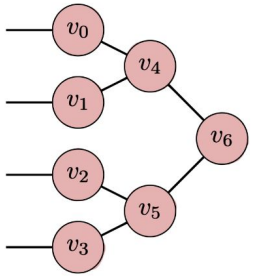
Matrix Product State (MPS)



Tree Tensor Network (TTN)







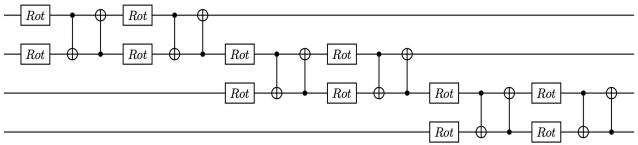


Figure 9. MPS meta-ansatz with two strongly entangling layers replacing each unitary tensor block.

Citation: Practical overview of image classification with tensor-network quantum circuits (Diego Guala, 2023)

Next steps: Other idea

- Possibly applying another classical optimizer? – (Other quasi-Newtonian methods)
- Exploring direct application:
  - Application of a variational hybrid quantum-classical algorithm to heat conduction equation. (Y. Y. Liu et al.)
  - A quantum algorithm for heat conduction with symmetrization. (S. Wei et al.)
- Barren Plateaus in this problem
- Other Discretization methods?

NNL Mini Apps
2nd Milestone Update

Mandy Bowman
Yury Chernyak
Horia Mărgărit

Goals for 2nd Milestone

Implement other ansatz

Investigate Barren Plateaus

Run code on Quantinuum devices

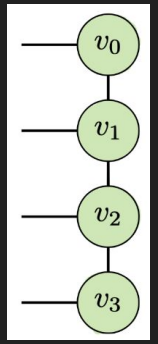
On Tensor Networks:

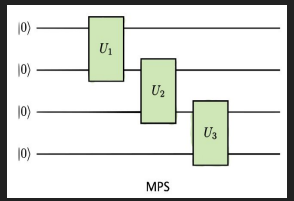
Tensor networks = are factorizations of large tensors into networks of smaller tensors

- A discretized function: function defined on some grid (taking a constant value on each grid cell
- Described as multi-dimensional array (i.e. a tensor)
- The connectivity of a tensor network is related to how entanglement is distributed
- A tensor network is a collection of tensors where a subset of all indices are contracted

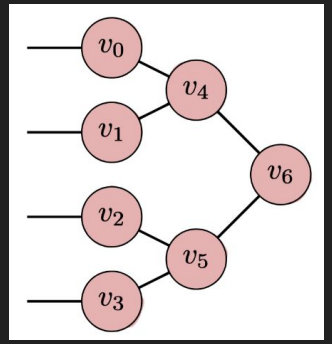
Two tensor network architectures are:

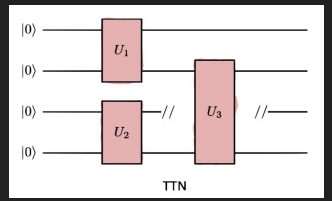
Matrix Product States (MPS)





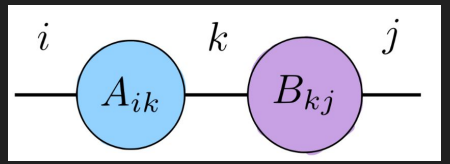
Tree Tensor Networks (TTN):





- **Rank**: num_of_indices in a tensor
  - [scalar = rank 0, vector = rank 1, matrix = rank 2]
- **Dimension**: num_of_elements that can be taken
  - [vect. w/ 3 elements has dim= 3]

Tensor Contraction:



$$C_{ij} = \sum_k A_{ik} B_{kj}$$

cont'd:

vector	v_j	
matrix	M_{ij}	
3-index tensor	T_{ijk}	

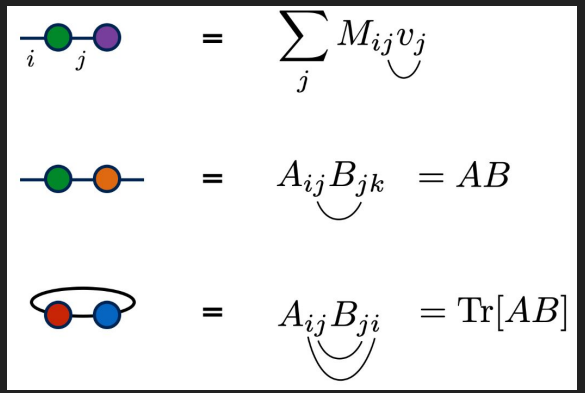
Notation: Math and Visual Representation:

https://tensornetwork.org/diagrams/

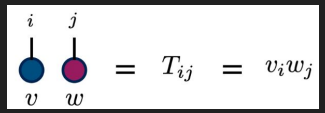
2 Main Rules for Diagrams:

1. Connecting 2 index lines => contraction or summation over connected indices
2. Tensor indices are noted by lines coming out of shapes–which are tensors

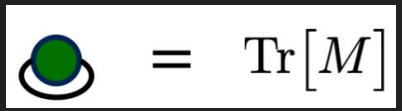
Examples of Matrix-like Contractions:



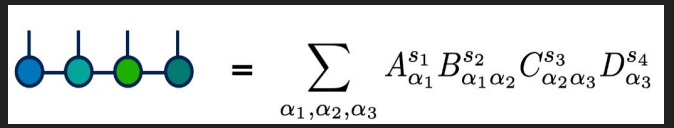
Outer Product:

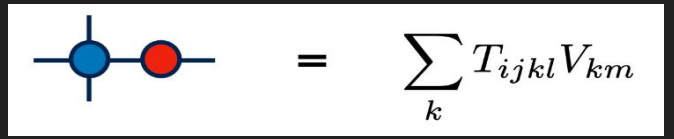


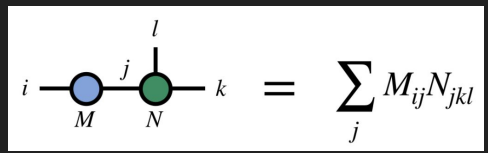
Trace:



Examples of Tensor Contractions:

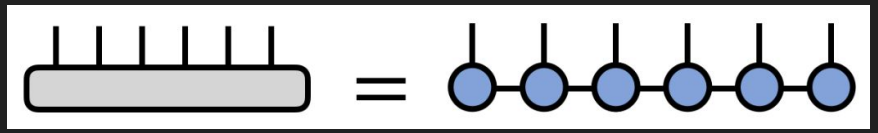






Matrix Product State/ Tensor Train

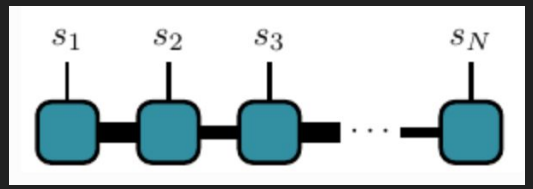
https://tensornetwork.org/mps/

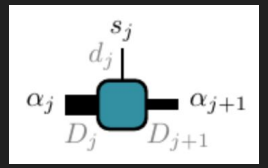


$$T^{s_1 s_2 s_3 s_4 s_5 s_6} = \sum_{\{\alpha\}} A^{s_1}_{\alpha_1} A^{s_2}_{\alpha_1 \alpha_2} A^{s_3}_{\alpha_2 \alpha_3} A^{s_4}_{\alpha_3 \alpha_4} A^{s_5}_{\alpha_4 \alpha_5} A^{s_6}_{\alpha_5}$$

- Special case of Tree Tensor Network
- Tensor networks extend matrix product state to higher dimensions
- Definition: *it's a factorization of a tensor with N indices into a chain-like product of 3-index tensors*

Vertical lines = physical indices
Horizontal lines = ancillary indices
Each square here represents a rank-3 tensor (A)





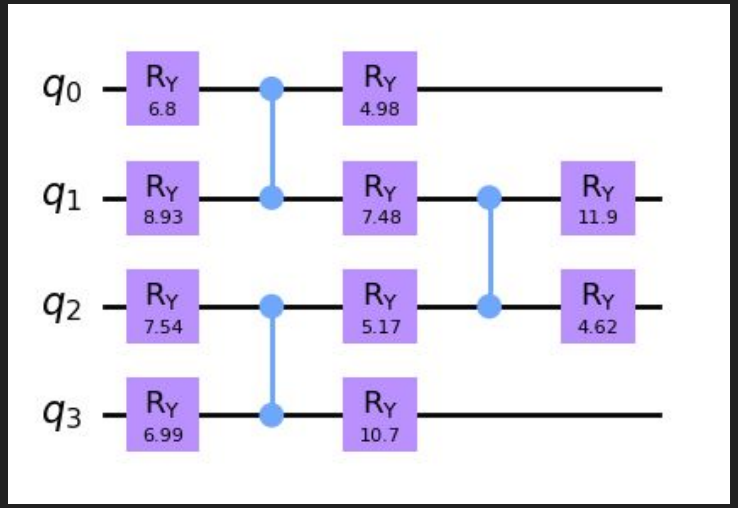
MPS of N particles

$$|\Psi\rangle = \sum_{\{s\}} \mathrm{Tr}\left[A_1^{(s_1)} A_2^{(s_2)} \cdots A_N^{(s_N)}\right] |s_1 s_2 \ldots s_N\rangle$$

On Tensor Networks:

- Our goal: translate the current 1D poisson function into tensor networks and represent it as such
- Our resources included:
  - Pennylane: https://pennylane.ai/qml/demos/tutorial_tn_circuits
  - Qiskit: https://github.com/Gopal-Dahale/ILearnQuantum/blob/main/tensor_networks_qiskit.ipynb
  - Other: https://tensornetwork.org/

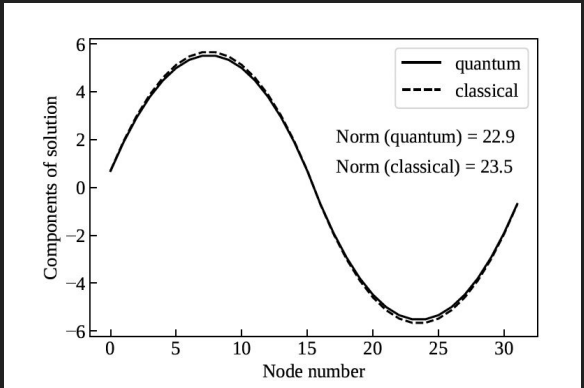
Original Ansatz: Hardware Efficient



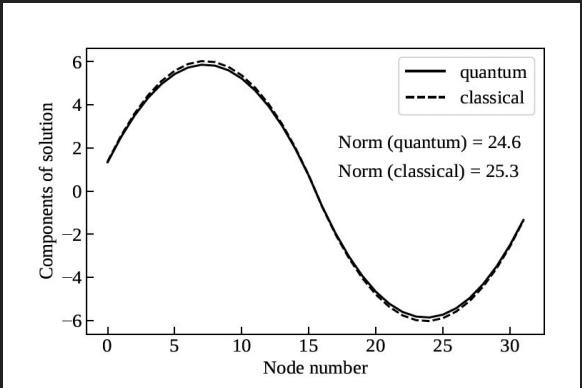
1 layer

Original Ansatz: Hardware Efficient

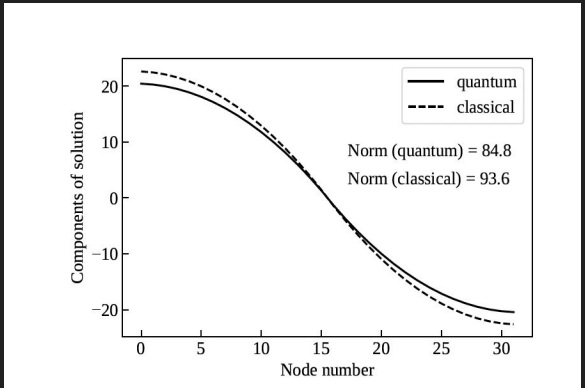
layers = 5
qubits = 5
trials = 10
Statevector Simulator



Periodic BC

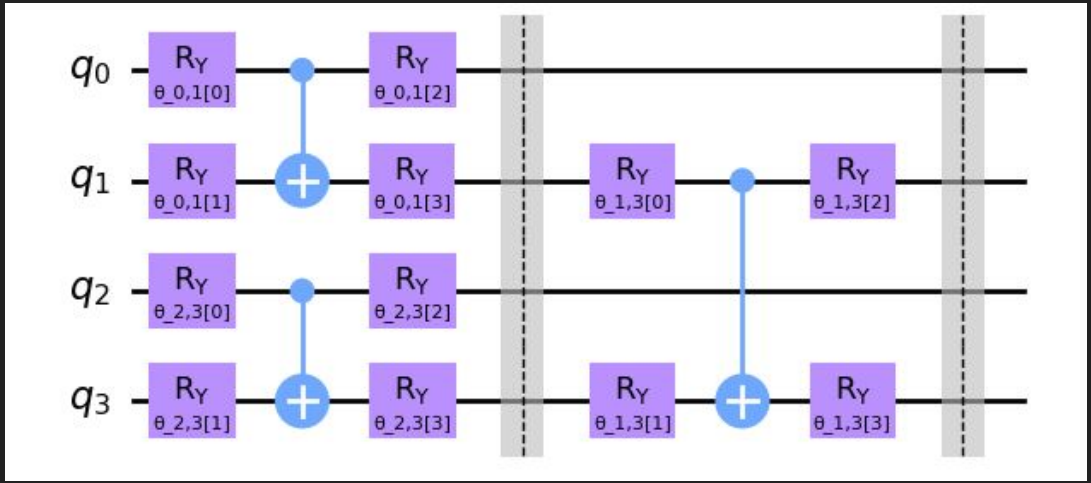


Dirichlet BC



Neumann BC

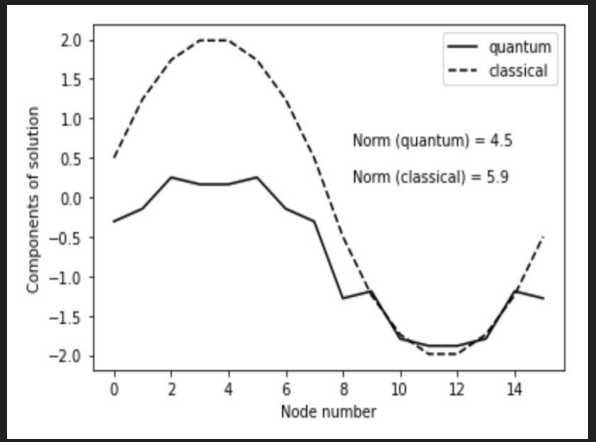
Other Ansatz: TTN



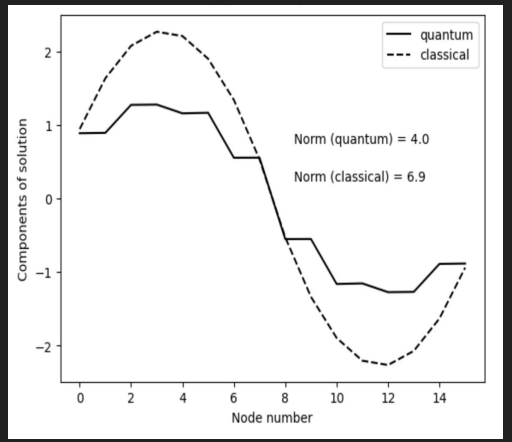
1 layer, 2 qubits per block

Other Ansatz: TTN

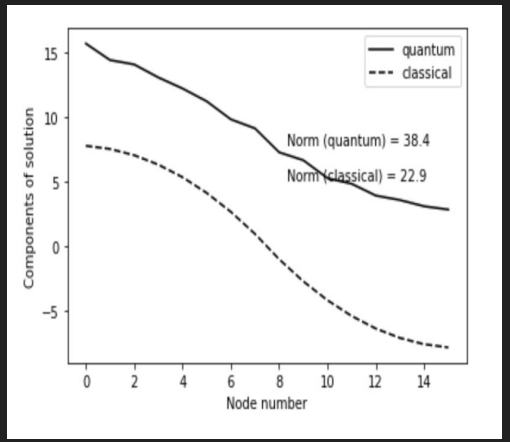
layers = 5
qubits = 5
trials = 10
Statevector Simulator



Periodic BC



Dirichlet BC

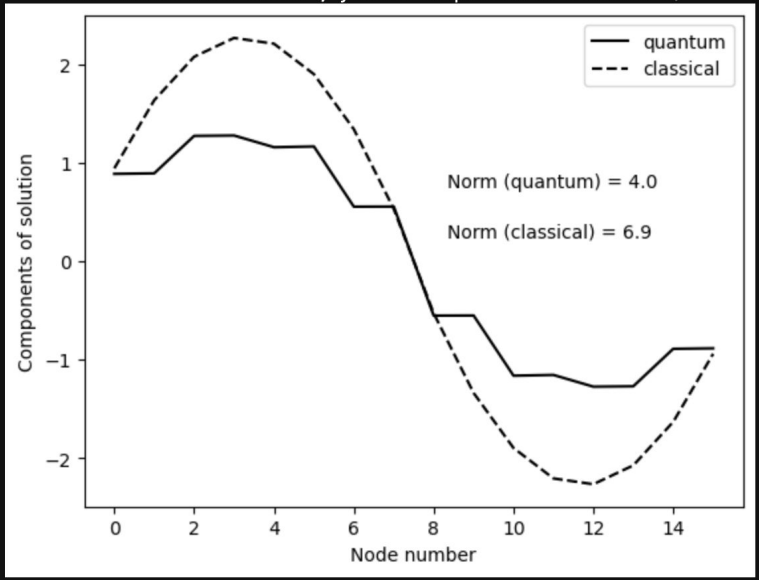


Neumann BC

TTN exhibits poor Bias-Variance tradeoff

```
In [9]:
print(bc, 'boundary condition, num_qubits:', data['num_qubits'][idx1])
q_sol = data['q_sol'][idx1][idx2]
cl_sol = data['cl_sol'][idx1]
plot_solution_vectors(q_sol, cl_sol)
```

```
Dirichlet boundary condition, num_qubits: 4
Out[9]: (<Figure size 640x480 with 1 Axes>,
         <Axes: xlabel='Node number', ylabel='Components of solution'>)
```




```
In [10]:
print('elapsed time: %.2e'%(time.time() - t0))
```

Contrasting `q_sol` to the *a priori* assumed ground truth, `cl_sol`, yields an aggregate measure of residual errors. These can be decomposed using the familiar bias-variance tradeoff.

MSE = Bias^2 + Variance

We plotted these two contrasting solutions using our custom Ansatz constructed using the design pattern of tree-tensor networks (TTNs)

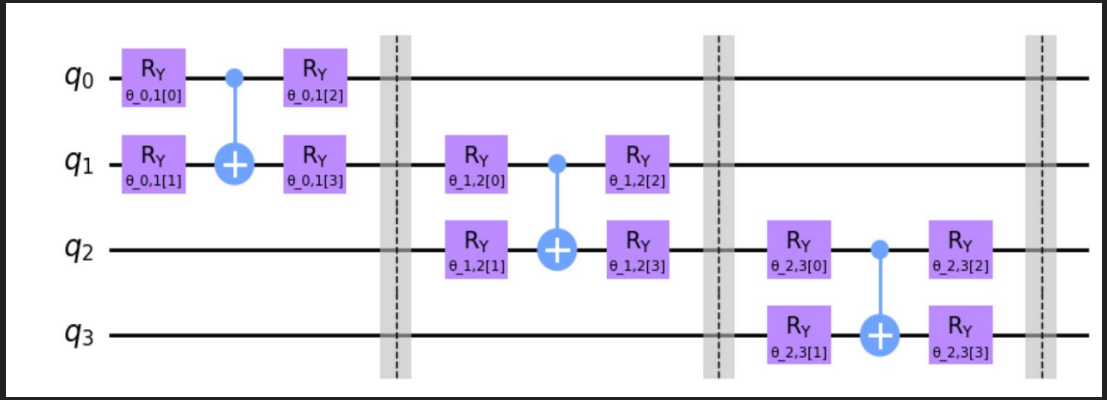
And we immediately observed that TTNs yield a `q_sol` with lower `Variance` but higher `Bias` than the *a priori* assumed ground truth `cl_sol`.

Shrinkage, as is purposefully designed in Lasso regression, is the likely explanation for this poor balance of the bias-variance tradeoff.

Modifying the TTN to have a branching factor of 3, is hypothesized to result in a better balance than our current TTN with a branching factor of 2. Because we view a TTN with a branching factor of 2 as equivalent to a L1 penalty on the `parameters` with a higher penalty coefficient / multiplier than having a TTN with a branching factor of 3. Put simply, we view a TTN as ***structural regularization*** not unlike Dropout from deep-learning literature.

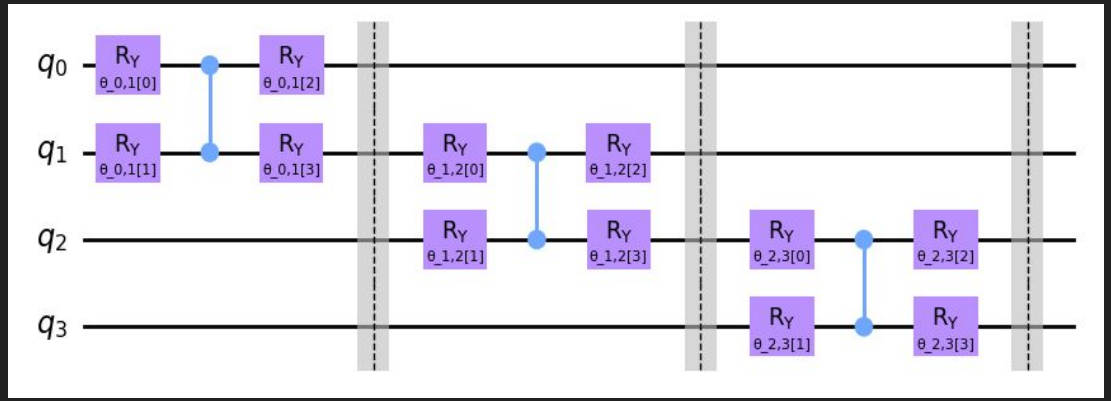
Specifically, we expect a TTN with a branching factor of 3 to exhibit less `Bias` but also less `Variance`. Because such a modified TTN would have less Shrinkage where ***we assume the number of parametrized gates to be a form of structural regularization*** not unlike the percent of nodes randomly omitted during deep-learning training when using Dropout.

Other Ansatz: MPS



1 layer, 2 qubits per block

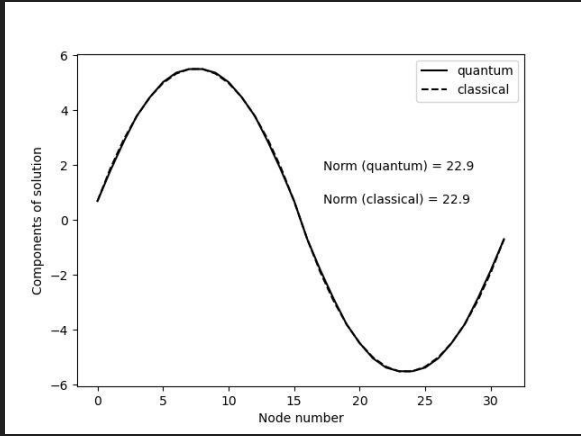
Other Ansatz: Custom MPS



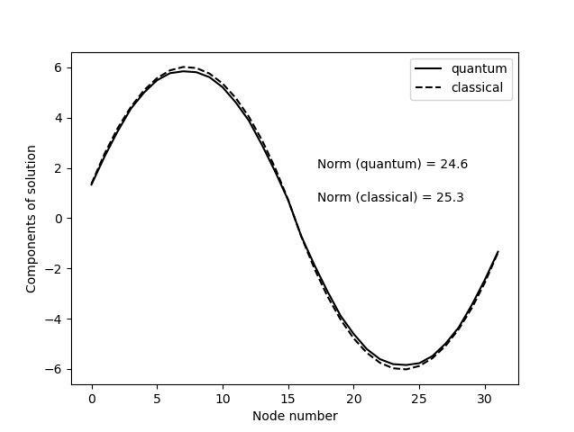
1 layer, 2 qubits per block
Changed Cx gate to Cz gate

Other Ansatz: Custom MPS

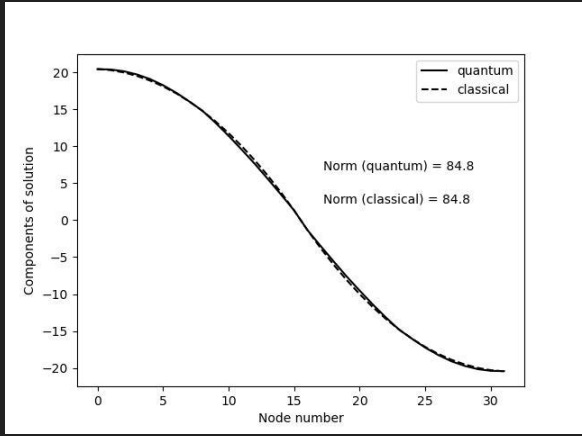
layers = 5
qubits = 5
trials = 10
Statevector Simulator



Periodic BC



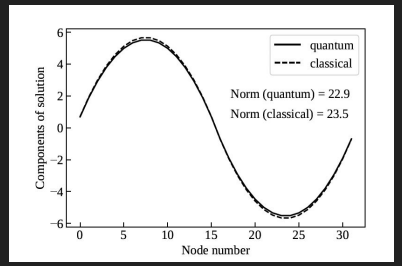
Dirichlet BC

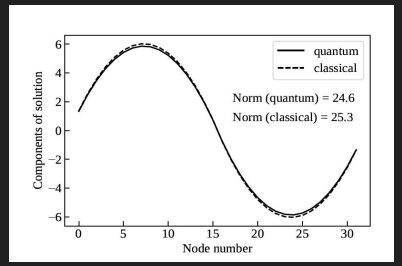


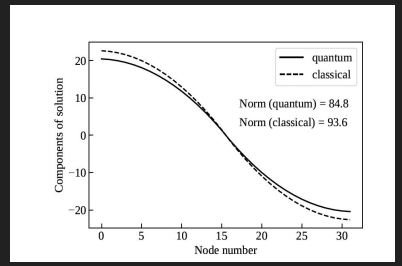
Neumann BC

Comparing Results

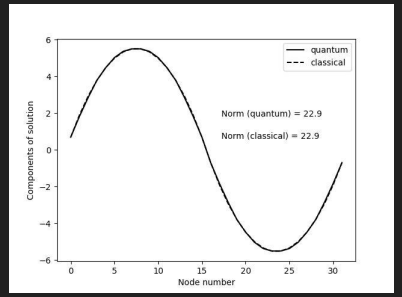
From Sato et al.

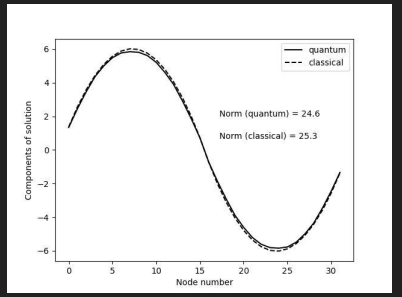


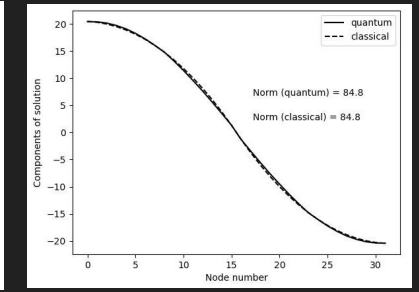




Custom MPS ansatz







Periodic

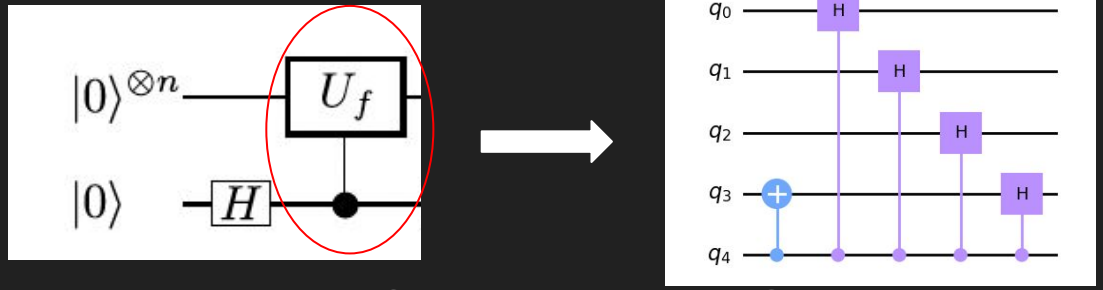
Dirichlet

Neumann

Running Code with Quantinuum

Straightforward to translate qiskit circuits to tket circuits using qiskit_to_tk()

Had to decompose Uf and U(theta) into individual control gates because you cannot translate a custom gate



Ultimately unable to run full simulation on Quantinuum emulator, H1-1E, because we ran out of credits

Next Steps:

More in depth analysis/evaluation of ansatz

Run on simulator that simulates a quantum computer (e.g. takes measurements)

Decide on new platform (qBraid?)

Run on a quantum computer

Next Steps:

When evaluating/ developing an ansatz, valuable questions include:

1. Will it improve time-complexity?
2. What is the expected accuracy of the solution?
3. Is it more suitable for expressing the solutions of certain PDE’s?
4. Will it avoid barren plateaus?
5. Does it minimize error?

NNL MINI APPS

Horia Mărgărit
Amanda Bowman
Yury Chernyak

Objective of NNL Mini-Apps Project

Find and implement quantum algorithms that solve common engineering problems

Objective of NNL Mini-Apps Project

Find and implement quantum algorithms that solve common engineering problems

Poisson equation

- Variety of engineering applications
- Recent publications for NISQ focused algorithms
- Available code bases

Evolution of Poisson Equation in Quantum Computing

2009 HHL linear solver algorithm is presented - requires fault-tolerance

2012 Cao - uses HHL in solving the Poisson Eqn

2019 Wang - uses HHL in solving the Poisson Eqn

2020 Lubasch - Variational Quantum Algorithms for nonlinear problems

2020 Lui - VQA to solve Poisson Eqn

2022 Sato - VQA to solve Poisson Eqn

NAVAL NUCLEAR LABORATORY

Sato et al. - Intro

$$E := \frac{1}{2}\int_\Omega \nabla v^* \cdot \nabla v \, d\Omega - \frac{1}{2}\int_\Omega v^* f \, d\Omega - \frac{1}{2}\int_\Omega f^* v \, d\Omega,$$

(ABOVE): Energy of system– physics motivated eqn.

Energy Minimization Method:

- Cost Function is based on the minimum P.E of a system

Boundary conditions considered:

- Periodic Boundary Condition
- Applying *Dirichlet* and *Neumann*— you get that the first and third terms become equal to zero and hence vanish $\Rightarrow$

$$\begin{aligned} -\boldsymbol{n}\cdot\nabla u(\boldsymbol{x}) &= 0 \quad \text{on} \quad \Gamma_{\mathrm{N}}, \\ u(\boldsymbol{x}) &= 0 \quad \text{on} \quad \Gamma_{\mathrm{D}}, \end{aligned}$$

Derivation:

$$\begin{aligned} 0 =& dE(v;\delta v) \\ =& \frac{1}{2}\int_\Omega \nabla \delta v^* \cdot \nabla v \, d\Omega + \frac{1}{2}\int_\Omega \nabla v^* \cdot \nabla \delta v \, d\Omega \\ & - \frac{1}{2}\int_\Omega \delta v^* f \, d\Omega - \frac{1}{2}\int_\Omega f^* \delta v \, d\Omega \\ =& \frac{1}{2}\int_{\Gamma_{\mathrm{N}}} \delta v^* \boldsymbol{n}\cdot\nabla v \, d\Gamma - \frac{1}{2}\int_\Omega \delta v^* \left(\nabla^2 v + f\right) \, d\Omega \\ & + \frac{1}{2}\int_{\Gamma_{\mathrm{N}}} \delta v \left(\boldsymbol{n}\cdot\nabla v\right)^* \, d\Gamma - \frac{1}{2}\int_\Omega \delta v \left(\nabla^2 v + f\right)^* \, d\Omega \end{aligned}$$

Conclusion:
minimizing the total potential energy w.r.t function v yields the state field u which is governed by poisson's equation.

$$-\nabla^2 u(\boldsymbol{x}) = f(\boldsymbol{x}) \quad \text{in} \quad \Omega.$$

Sato et al. - Method Part 1

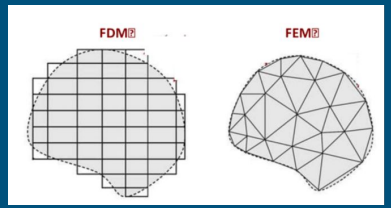
Poisson Eqn s.t *x* is defined on *d*-dimensional cubic domain:

- Solving = *discretizing* the equation => matrix representing the system (Quantum? perhaps…)
- Decompose matrix and map components into quantum states & onto quantum circuit.

Discretization:

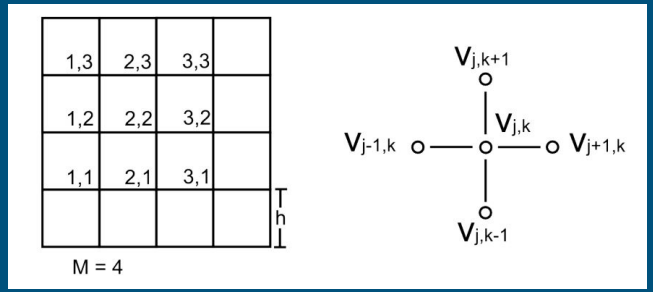
Finite Element Method (FEM)

Finite Difference Method (FDM)



Discretization Function:

- *defined over some grid, taking constant value on each cell*
- *Can be described by a multidimensional array (vector, tensor)*



Sato et. al. - Methods Part 2

- Taking 2 neighboring points, subtract one from other, d**erivative**.
- Applying twice you get **2nd-order**.
- Poisson's Eqn uses 2nd-order derivative
- The Discretized Function can be represented as the second order derivative in vector form

$$f'(x) \approx \frac{f(x+\delta x) - f(x)}{\delta x}$$

$$f''(x) \approx \frac{f(x+\delta x) - 2f(x) + f(x-\delta x)}{\delta x^2}$$

$$\frac{1}{\delta x^2}\begin{pmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_{i+1}) \\ \vdots \\ f(x_N) \\ f(x_0) \end{pmatrix} - \frac{2}{\delta x^2}\begin{pmatrix} f(x_0) \\ f(x_1) \\ \vdots \\ f(x_i) \\ \vdots \\ f(x_{N-1}) \\ f(x_N) \end{pmatrix} + \frac{1}{\delta x^2}\begin{pmatrix} f(x_N) \\ f(x_0) \\ \vdots \\ f(x_{i-1}) \\ \vdots \\ f(x_{N-2}) \\ f(x_{N-1}) \end{pmatrix}$$

$$h^{-2}\left((-v_{j-1,k} + 2v_{j,k} - v_{j+1,k}) + (-v_{j,k-1} + 2v_{j,k} - v_{j,k+1})\right) = f_{j,k}$$

$$-\Delta_h \vec{v} = \vec{f}_h$$

Poisson's Eqn can be recast as linear system: ***Au = f***

Sato et. al. - Methods Part 3

Results/Set-Up from Sato:

$$A_{\text{periodic}} = I^{\otimes n-1} \otimes (I - X) + P^{-1} \left(I^{\otimes n-1} \otimes (I - X)\right) P \qquad P := \sum_{i \in [0, 2^n - 1]} |(i+1) \bmod 2^n\rangle \langle i|$$

$$A_{\text{Dirichlet}} = A_{\text{periodic}} + P^{-1} \left(I_0^{\otimes n-1} \otimes X\right) P$$

$$A_{\text{Neumann}} = A_{\text{periodic}} - P^{-1} \left(I_0^{\otimes n-1} \otimes (I - X)\right) P.$$

Periodic:

$$\begin{bmatrix} 2 & -1 & 0 & & \dots & & 0 & -1 \\ -1 & 2 & -1 & 0 & \dots & & & 0 \\ 0 & -1 & 2 & -1 & \dots & & & 0 \\ \vdots & & & \ddots & & & & \vdots \\ 0 & & & \dots & 0 & -1 & 2 & -1 \\ -1 & 0 & & \dots & & 0 & -1 & 2 \end{bmatrix}$$

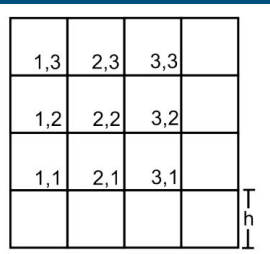
Neumann:

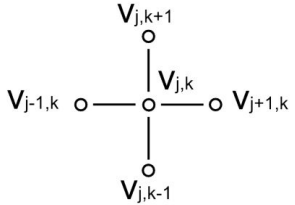
$$\begin{bmatrix} 1 & -1 & 0 & & \dots & & & 0 \\ -1 & 2 & -1 & 0 & \dots & & & 0 \\ 0 & -1 & 2 & -1 & \dots & & & 0 \\ \vdots & & & \ddots & & & & \vdots \\ 0 & & & \dots & 0 & -1 & 2 & -1 \\ 0 & & & \dots & & 0 & -1 & 1 \end{bmatrix}$$

Dirichlet:

$$\begin{bmatrix} 2 & -1 & 0 & & \dots & & & 0 \\ -1 & 2 & -1 & 0 & \dots & & & 0 \\ 0 & -1 & 2 & -1 & \dots & & & 0 \\ \vdots & & & \ddots & & & & \vdots \\ 0 & & & \dots & 0 & -1 & 2 & -1 \\ 0 & & & \dots & & 0 & -1 & 2 \end{bmatrix}$$

$$A = L_h \otimes I + I \otimes L_h$$





$$h^{-2} A \begin{pmatrix} v_1 \\ \vdots \\ v_9 \end{pmatrix} := h^{-2} \begin{pmatrix} B & -I & \\ -I & B & -I \\ & -I & B \end{pmatrix} \begin{pmatrix} v_1 \\ \vdots \\ v_9 \end{pmatrix} = \begin{pmatrix} f_1 \\ \vdots \\ f_9 \end{pmatrix}$$

$$-\Delta_h = h^{-2} A$$

$h = 1$ *(in this case)* hence form is $\boldsymbol{Av} = \boldsymbol{f}$

$$-\Delta_h \vec{v} = \vec{f}_h \quad \Longleftrightarrow \quad -\nabla^2 u(\boldsymbol{x}) = f(\boldsymbol{x})$$

$$A = \begin{pmatrix} L_h + 2I & -I & 0 & \cdots & \cdots & 0 \\ -I & L_h + 2I & -I & 0 & \cdots & 0 \\ 0 & -I & \ddots & \ddots & 0 & \vdots \\ \vdots & 0 & \ddots & \ddots & -I & 0 \\ \vdots & \vdots & 0 & -I & L_h + 2I & -I \\ 0 & 0 & \cdots & 0 & -I & L_h + 2I \end{pmatrix}$$

NAVAL NUCLEAR LABORATORY

Quantum Version of Problem

After Discretization:

$$E_h := \frac{1}{2}\boldsymbol{v}^* \cdot A\boldsymbol{v} - \frac{1}{2}\boldsymbol{v}^* \cdot \boldsymbol{f} - \frac{1}{2}\boldsymbol{f}^* \cdot \boldsymbol{v},$$

- *v* and *f* denote vectors with component values of v and f at the nodes discretizing the domain Ω

After doing the following:

- ➢ encoding *f* and *v* into quantum states with parameterized solution state
- ➢ Preparing *f* as $|f, \psi(\theta)\rangle := \sqrt{}(|0\rangle |f\rangle + |1\rangle |\psi(\theta)\rangle) / 2$
- ➢ applying the necessary condition for optimality (requiring partial of $Eh(r,\theta)$ w.r.t *r* is equal to zero)

Cost Function to Optimize:

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

Note: different **A***'s for different boundary conditions*

Comparing to Liu Paper:

$$E(\boldsymbol{\theta}) = \langle \psi(\boldsymbol{\theta}) \,|\, A\,(I - |f\rangle\langle f|)\,A \,|\, \psi(\boldsymbol{\theta})\rangle$$

NAVAL NUCLEAR LABORATORY

Sato et al. - Limitations

- Hardware efficient ansatz suffers from barren plateaus at long depths
- Gradient-based optimizers lead to deep circuits
  - # of circuits is proportional to # of parameters
  - # of iteration is dependent on the classical optimizer

	Xn	Xn w/shift add	Gradient of A	Gradient of A w/shift add	X0
# of gates	37	85	256	304	134
# of iterations	121	121	736	736	857

4 qubits, 2 layers, Hardware Efficient Ansatz, Periodic BCs

NAVAL NUCLEAR LABORATORY

Our Goals

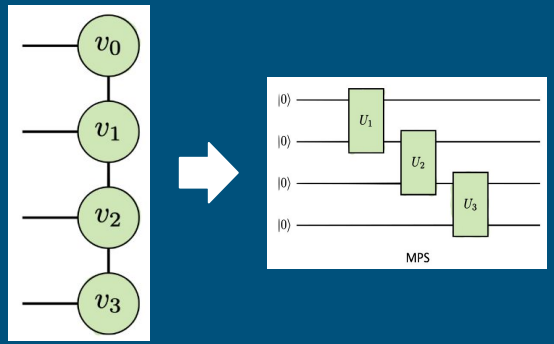
1. Implement different ansatz
2. Modify classical optimizers
3. Run on simulators
4. Run on quantum hardware

NAVAL NUCLEAR LABORATORY

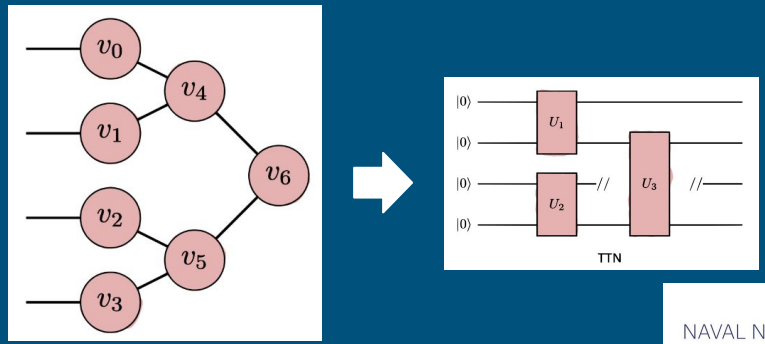
Implementing different ansatz: Tensor Networks

- Tensor networks are factorizations of large tensors into networks of smaller tensors
- Described as multi-dimensional array (i.e. a tensor)
- The connectivity of a tensor network is related to how entanglement is distributed

Matrix Product States (MPS)

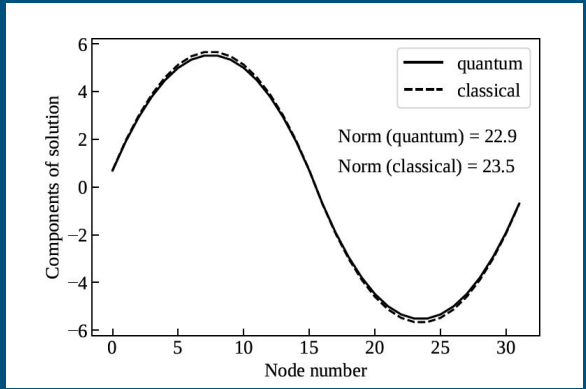


Tree Tensor Networks (TTN):



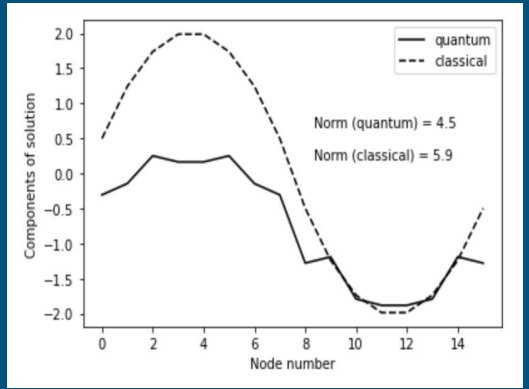
Preliminary Results: IBM Statevector Simulator

Periodic BCs
layers = 5
qubits = 5



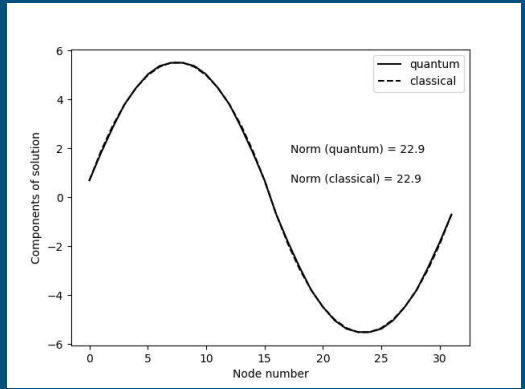
From Sato et al. - hardware efficient ansatz

Periodic BCs
layers = 5
qubits = 4



TTN Ansatz

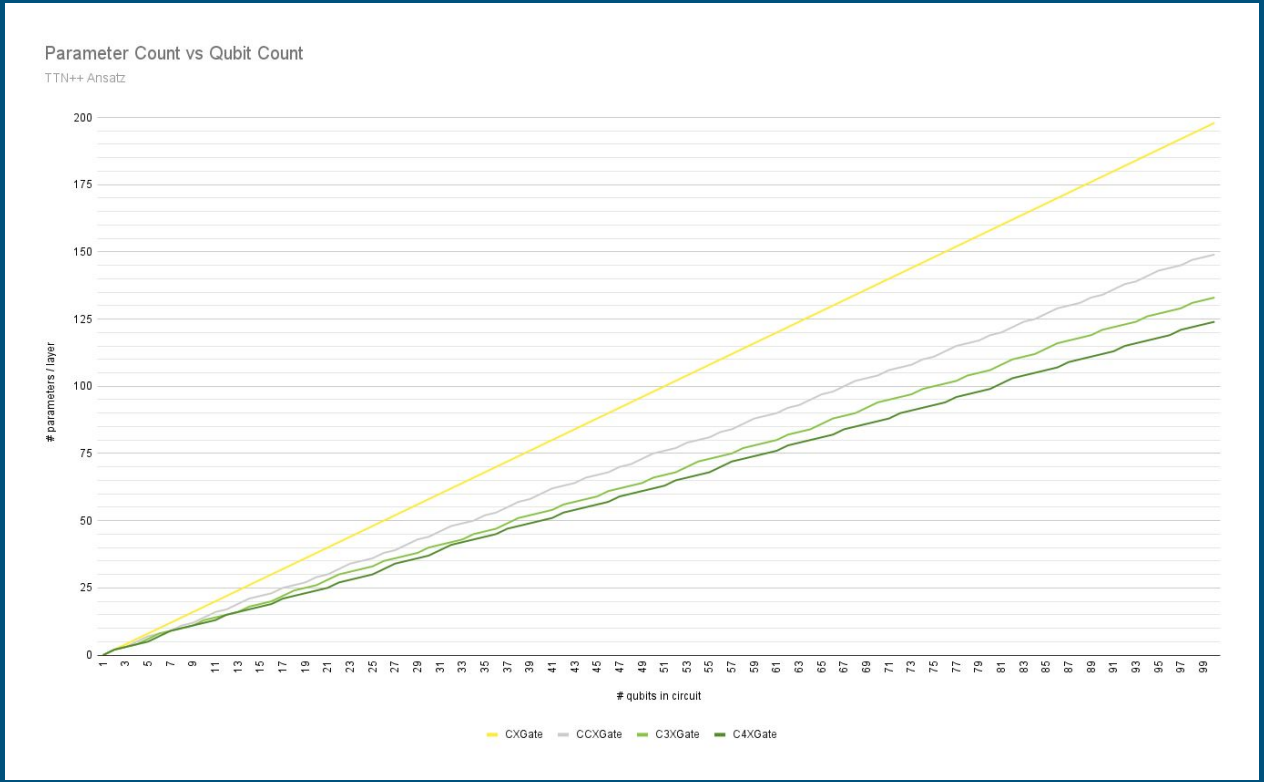
Periodic BCs
layers = 5
qubits = 5



MPS Ansatz

NAVAL NUCLEAR LABORATORY

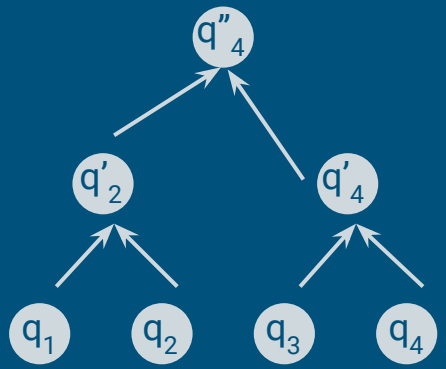
Tensor Networks: Improving time complexity



Tensor Networks: Improving time complexity

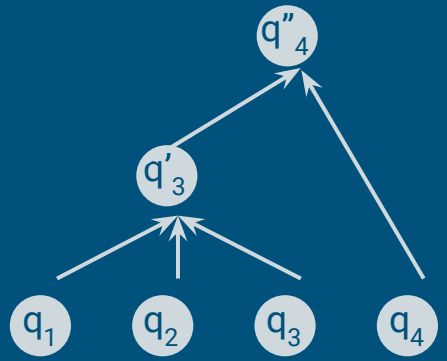
c.f., our github repository

TTN
(4 qubits)



From:
- 9 gates / layer
- 6 parameters / layer

TTN++
(4 qubits)



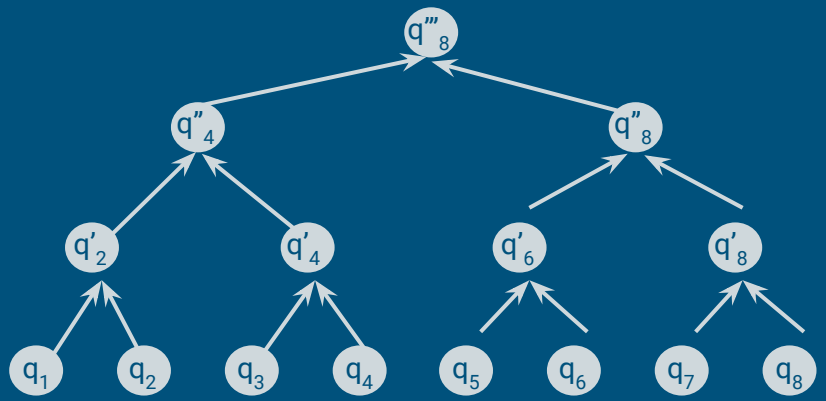
To:
- 7 gates / layer
- 5 parameters / layer

NAVAL NUCLEAR LABORATORY

Tensor Networks: Improving time complexity

c.f., our github repository

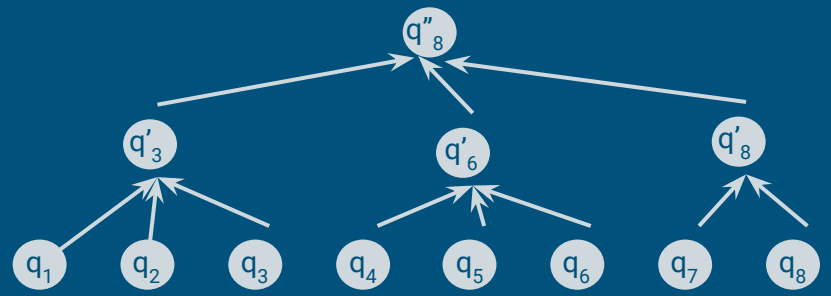
TTN (8 qubits)



From:

- 21 gates / layer
- 14 parameters / layer

TTN++ (8 qubits)



To:

- 15 gates / layer
- 11 parameters / layer

NAVAL NUCLEAR LABORATORY

Tensor Networks: Escaping barren plateaus

Minimizing $|| f_q(\mathbf{x};\boldsymbol{\theta}) - f_{cl}(\mathbf{x};\boldsymbol{\phi}^*) ||_2 + \lambda || \boldsymbol{\theta} ||_1$ provably minimizes $MSE(q\_sol, cl\_sol^*)$ subject to making $\boldsymbol{\theta}$ sparse, by shrinking some entries to 0 through iterative decrements by λ.

Similar parameter sparsity can be imposed *structurally* as is done by *reducing the number of parametrized gates* in our tensor networks. By entangling more qubits with Toffoli gates, we reduce the number of parametrized gates from 18 to 14 per layer, for an ansatz using 10 qubits. The new ansatz is 22% more sparse, yet still required to span the solution space to the Poisson equation.

Structural sparsity leads to a compressed representation of the solution space. Shrinking the barren plateaus and helping to escape them.

Let $q\_sol \equiv f_q(\mathbf{x};\boldsymbol{\theta})$ and $cl\_sol \equiv f_{cl}(\mathbf{x};\boldsymbol{\phi})$. Given $\boldsymbol{\phi}^*$ such that its corresponding cl_sol^* is optimal. Find $\boldsymbol{\theta}^*$ and its q_sol^* by minimizing $|| f_q(\mathbf{x};\boldsymbol{\theta}) - f_{cl}(\mathbf{x};\boldsymbol{\phi}^*) ||_2 + \lambda || \boldsymbol{\theta} ||_p$ where $\lambda \geq 0$ and $p \geq 1$.

Recall that $|| \boldsymbol{\theta} ||_p = \left(\sum_i | \theta_i |^p \right)^{\frac{1}{p}}$ and suppose we choose $p = 1$, then $\lambda || \boldsymbol{\theta} ||_1 = \lambda \sum_i | \theta_i |$ and $\nabla_{\boldsymbol{\theta}} \lambda || \boldsymbol{\theta} ||_1 = \lambda \, sign(\boldsymbol{\theta})$ which simply increments / decrements $\boldsymbol{\theta}$ by λ during the minimization.

NAVAL NUCLEAR LABORATORY

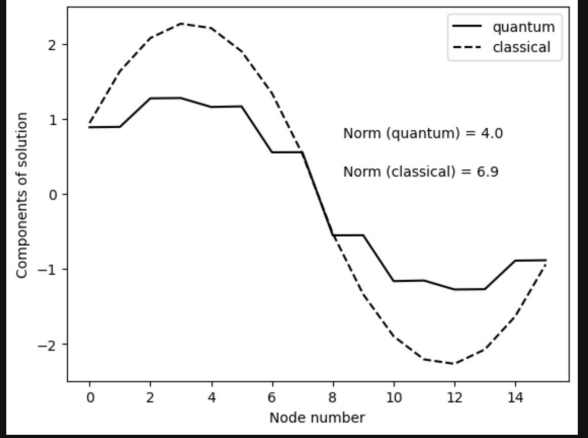
Tensor Networks:
Structural regularization and shrinkage

We observed that TTNs with *at most 2-qubit entanglement* yield a q_sol with lower Variance but higher Bias than the *a priori* assumed ground truth cl_sol.

$$
\begin{aligned}
&MSE\left(cl\_sol, q\_sol\right) \\
&= \left(\mathbb{E}\left[q\_sol\right] - cl\_sol\right)^2 + \mathbb{E}\left[\left(q\_sol - \mathbb{E}\left[q\_sol\right]\right)^2\right] + \epsilon \\
&= Bias^2 + Variance + Irreducible\ Error
\end{aligned}
$$

```
In [9]: print(bc, 'boundary condition, num_qubits:', data['num_qubits'][idx1])
        q_sol = data['q_sol'][idx1][idx2]
        cl_sol = data['cl_sol'][idx1]
        plot_solution_vectors(q_sol, cl_sol)

        Dirichlet boundary condition, num_qubits: 4
Out[9]: (<Figure size 640x480 with 1 Axes>,
          <Axes: xlabel='Node number', ylabel='Components of solution'>)
```



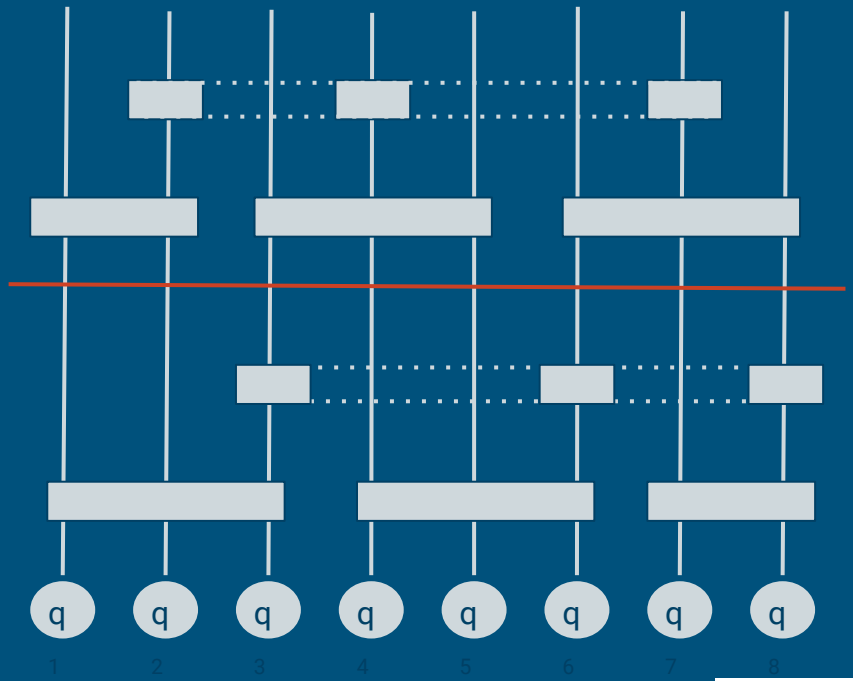

```
In [10]: print('elapsed time: %.2e'%(time.time() - t0))
```

Tensor Networks:
Improved bias / variance tradeoff

Shrinkage, as is purposefully designed in Lasso regression, is the likely explanation for this poor balance of the bias-variance tradeoff.

We need to *counterbalance* the reduced number of parametrized gates *with more layers of the ansatz*.

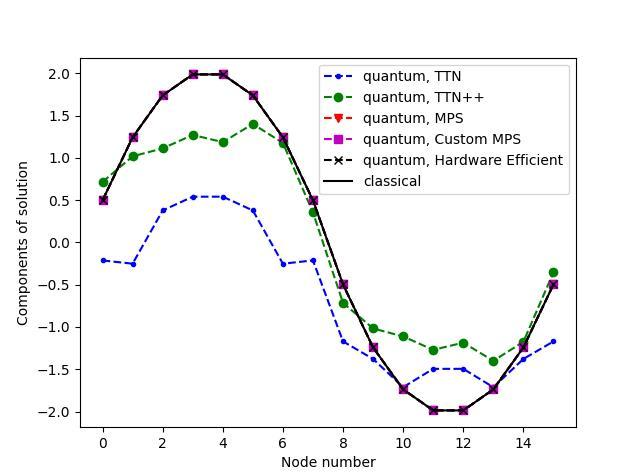
Future implementations should also *randomize the entanglement groups* from one layer to another.



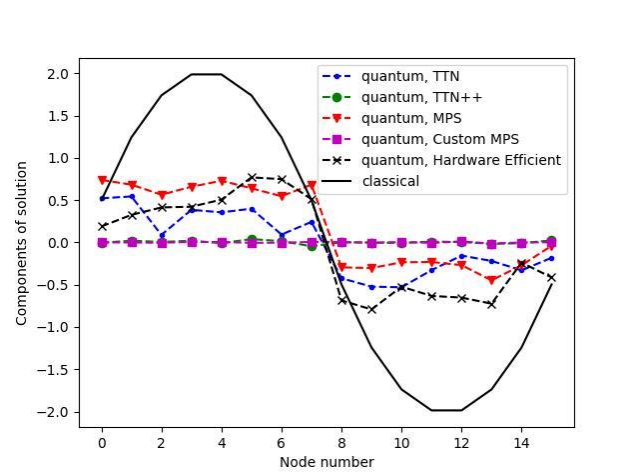
NAVAL NUCLEAR LABORATORY

More Results

IBM Statevector Simulator (Matrix-Vector Multiplication)



IBM QASM Simulator (Ideal Quantum Computer)



More Results

IBM Statevector Simulator (Matrix-Vector Multiplication)

TTN	179 s
TTN++	82 s
MPS	183 s
Custom MPS	167 s
Hardware Efficient	114 s

IBM QASM Simulator (Ideal Quantum Computer)

TTN	107 s
TTN++	9 s
MPS	103 s
Custom MPS	43 s
Hardware Efficient	38 s

NAVAL NUCLEAR LABORATORY

What can be said about our tensor network ansatze?

c.f., our github repository

- Will they improve time-complexity? Yes
- What is the expected accuracy of the solution? Unknown
- Is it more suitable for expressing the solutions of certain PDE's? Unknown
- Will it escape barren plateaus? Yes
- Can their error be minimized? Yes

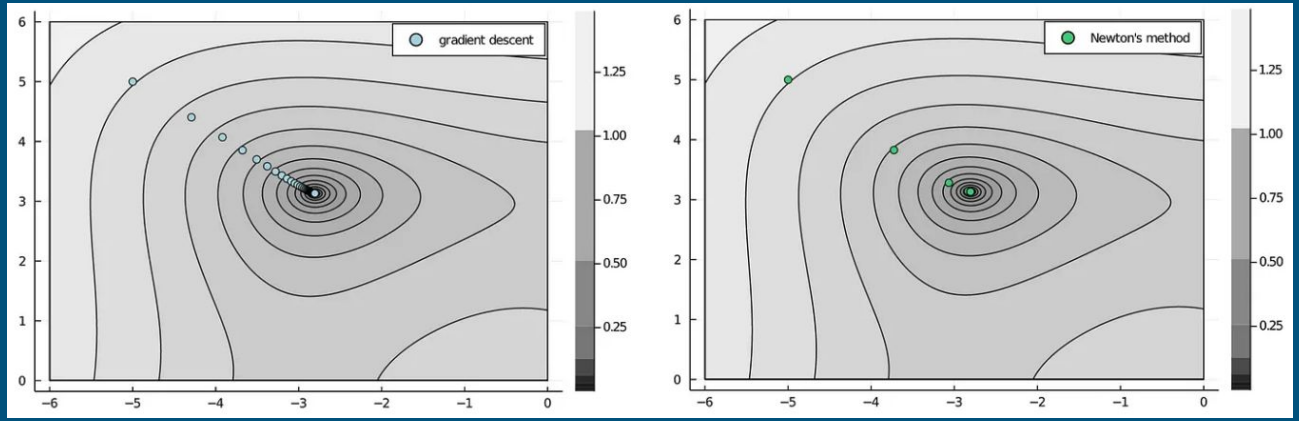
CLASSICAL OPTIMIZERS:

Gradient Descent: $\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha \nabla f(\mathbf{x_k})$

Newton's Method: $\mathbf{x}_{k+1} = \mathbf{x}_k - H(\mathbf{x}_k)^{-1} \nabla f(\mathbf{x}_k)$

$$H_f = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \, \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \, \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \, \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \, \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \, \partial x_1} & \frac{\partial^2 f}{\partial x_n \, \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

Comparison:



Classic Optimizers used in the Paper - BFGS

- BFGS = *quasi-Newton* optimization method — i.e Gradient based for smooth unconstrained non-linear objective functions w/out **HESSIAN**
- Approximates Hessian with a positive-definite matrix
- One precondition must be satisfied ("Secant Method"):

$$B_{k+1}[\mathbf{x}_{k+1} - \mathbf{x}_k] = \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)$$

Minimizing $Ax = b$ can be equivalent to finding minimum of quadratic form:

Start w/ a guess *x0*:

$$f(x) = \frac{1}{2}x^T Ax - b^T x$$

& compute residual:

$$r_i = b - Ax_i$$

Minimize each iter w/ a line search:

$$x_{i+1} = x + \alpha_i r_i$$

with:

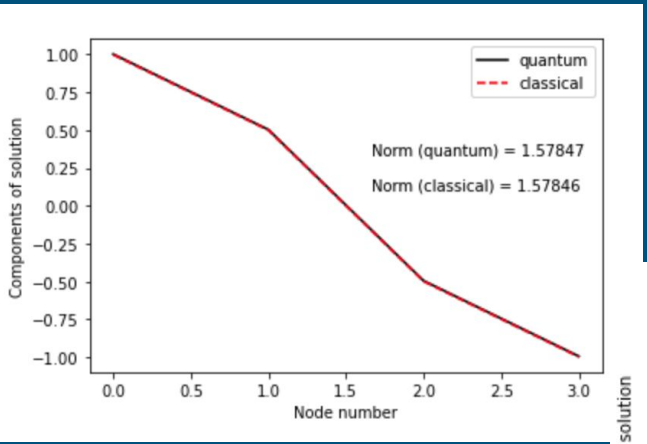
$$\alpha_i = \frac{r_i{}^T r_i}{r_i{}^T A r_i}$$

NAVAL NUCLEAR LABORATORY

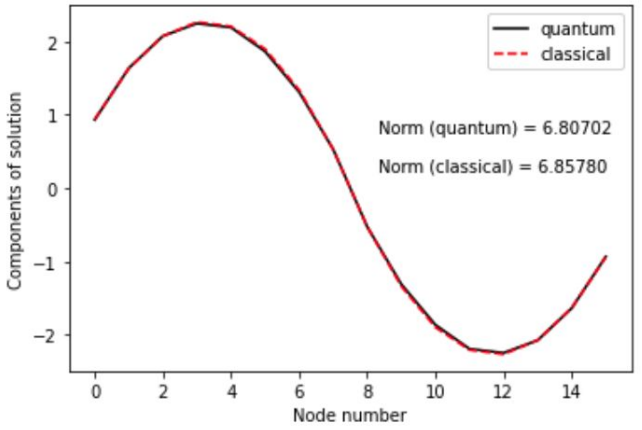
Graphs for CG Optimizer:

- *2 qubits*
- *Custom MPS*
- *Layers=5*

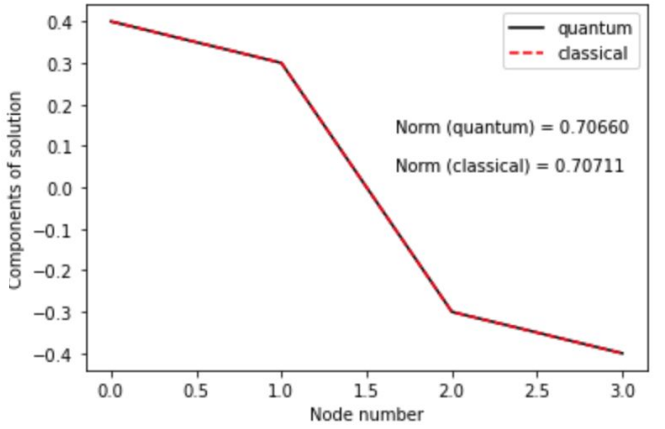
Periodic BC:



Dirichlet BC:



Neumann BC:



Running on Quantum Hardware

- Had access to Quantinuum credits – simulation required far too many resources and we quickly ran out of credits
- Applied to UMD proposal to run on quantum hardware – did a deeper analysis of the resource requirements – helped us identify where the algorithm could be optimized
  - Stick with periodic bcs
  - Modify classical optimizer
  - Try different ansatz

Future Work: Phase 2

- Implement noise models
- Error correction and optimization
- Run on simulators with noise models and possibly run on real quantum hardware
- Hear back from UMD → decide on which platform we want to continue with (noise models and error correction methods we use will depend on which platform we use)