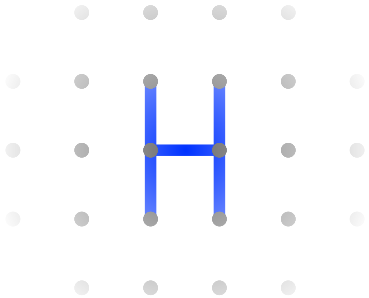
Supplementary material to 'Variational quantum eigensolver for the Heisenberg antiferromagnet on the kagome lattice'

Joris Kattemölle[1,2,*] and Jasper van Wezel[1,2]

[1] *Institute for Theoretical Physics, University of Amsterdam, Science Park 904, Amsterdam, The Netherlands*
[2] *QuSoft, CWI, Science Park 123, Amsterdam, The Netherlands*

Here, we introduce the code used to generate the data in the main text. The file SM.zip contains the code. Further documentation, with plenty of examples, is embedded into the code by docstrings. This introduction is adapted from Ref. [**?**].



1. Introduction

`HeisenbergVQE` is a python package tailored to Variational Quantum Eigensolvers for the Heisenberg model on any graph, with optional GPU acceleration. Let us first introduce `HeisenbergVQE` by an example. Say you want to use a VQE (using the Hamiltonian variational ansatz, see the main text) to find the ground state of the Heisenberg model on a square. In this example we will assume HeisenbergVQE is installed into `~/HVQE` (see Sec. **??** for installation instructions). Create a text file with the contents

```
complete_graph_input = [(0,1),(1,2),(2,3),(3,0)]
init_layer_input = [(0,1),(2,3)]
layers_input = [
  [(1,2),(3,0)],
  [(0,1),(2,3)]
  ]
```

and save it under `~/HVQE/graph_input.txt`. Here:

- The list of edges `complete_graph_input` defines the graph the Hamiltonian is defined on by defining its edges, using integers as vertices.
- The list of edges `init_layer_input` defines the initial state of the register by specifying the edges along which singlets are to be created.
- The list of lists of edges `layers_input` defines the cycle of the ansatz. The l'th list contains the edges along which Heisenberg gates $e^{-i\alpha(XX+YY+ZZ)/4}$ are to placed in the l'th layer of the cycle. Here, α is the parameter of the gate, and XX stands for the tensor product of two Pauli-X operators (likewise for YY and ZZ).

To be able to assess the performance of the VQE, first compute the ground state and the first excited state by exact diagonalization. Enter the following command in a Unix shell.

```
\$ python3 ~/HVQE/ground_state.py ~/HVQE 2
```

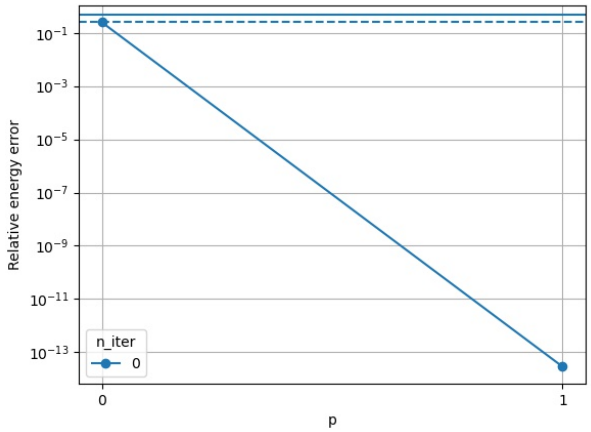
The script `~/HVQE/ground_state.py` looks for the file `graph_input.txt` inside the folder `~/HVQE`. It then computes the ground state of the `2` lowest energy eigenstates and stores results in `~/HVQE/lowest_energies.txt` (listing the `2` lowest energies) and `~/HVQE/gs.dat` (a data file containing the ground state itself).

Now we can run the VQE sequentially using `0` parameters (i.e. just return the energy of the init state) and `4` parameters, with a parameter multiplicity of `1` (i.e. every Heisenberg gate gets its own parameter).

```
\$ python3 ~/HVQE/HVQE.py ~/HVQE 0 1
\$ python3 ~/HVQE/HVQE.py ~/HVQE 4 1
```

(For detailed info about the command line options, run `\$ python3 ~/HVQE/HVQE.py -h`.)

The above commands run the VQE with the ansatz specified in `~/HVQE/graph_input.txt`, and the following semilog plot of the result is saved under `~/HVQE/E_VQE.pdf`.



* Present address: University of Konstanz, Germany

The vertical axis displays the relative error to the ground state $\mathcal{E}$, as introduced in the main text. The horizontal axis displays the number of cycles used in the ansatz. The solid horizontal line is at the energy of the first excited state. The dashed horizontal line is halfway between the first exited state and the ground state. (The ground state itself is not shown because it is at $\mathcal{E} = 0$.)

We see our simple VQE has already found the exact ground state of the Heisenberg model on the square after a single cycle of the ansatz (a circuit of depth 2, excluding singlet preparation).

2. Files and folders

For usage of HeisenbergVQE, the most important files and folders are:

- `HVQE.py`. Call this scipt to run a VQE. For detailed info on its command-line usage, run `\$ python3 HVQE.py -h`.
- `qem.py`. (Quantum EMulator) This script defines standard functions and classes (e.g. gates, circuits, circuit execution) needed for the emulation of quantum circuits. Matrix-vector multiplication is handled by NumPy if no CuPy installation is found. Otherwise it is handled by CuPy (i.e. on a CUDA-enabled GPU, if available).
- `ground_state.py`. Usage: `\$ python3 ground_state.py path k`. Compute the `k` lowest energy eigenstates of the Heisenberg model on the graph defined in `path/graph_input.txt`. The (flattened) ground state is saved under `path/gs.dat`, and a list of the `k` lowest energies is stored under `path/lowest_energies.txt`.
- `noise.py` Script used for obtaining the noisy data. Run `\$ python3 noise.py -h` for command line details.
- `data/` Contains pre-produced data that was obtained by running `HVQE.py`. The folder contains a tree of folders, where every leaf contains all input and output pertaining to a single, specific system and ansatz. The data pertaining to the plots in the main text is stored in the following sub-folders.
  - `./kagome/open/on_grid/opg/energy/2x2/` This is the data for the plots on KVQE_G, that is, the `kagome` lattice with `open` boundary conditions, compiled for a processor with grid connectivity (`on_grid`), using one parameter per gate (`opg`), using the `energy` as a cost function, for a patch of `2x2` unit cells.
  - `./kagome/periodic/own_graph/opg/energy/3x2/` This is the data for the plots on KVQE_K, that is, the `kagome` lattice with `periodic` boundary conditions, compiled for a processor with the connectivity of the periodic kagome lattice itself (`own_graph`), using one parameter per gate (`opg`), using the `energy` as a cost function, for a patch of `3x2` unit cells.
  - `./chain/periodic/own_graph/opg/energy/20/` This is the data for the plots on CVQE, that is, the `chain` with `periodic` boundary conditions, compiled for a processor with the connectivity of the periodic chain itself (`own_graph`), using one parameter per gate (`opg`), using the `energy` as a cost function, for a patch of `3x2` unit cells. The data pertaining to the system-size scaling in the main text is found by replacing `20/` by other system sizes, ranging from 4 to 22 in steps of 2. (No data needed to generated for $n = 2$ because in that case the initial state is the ground state.)
- `analysis/` Contains the scripts used the generate the plots in the main text.

3. Installation

1. Download and unpack HVQE.zip. If you have git installed, you can clone HVQEs online repository by `\$ git clone https://github.com/barbireau/HVQE.git` instead.
2. Install Miniconda [**?**] or Anaconda [**?**]. If you have a CUDA-enabled GPU and you want to use GPU acceleration, `cd` to `~/HVQE` and run `\$ conda env create -f environment/environment_GPU.yml` to set up the right python environment. If you do not have such a GPU, or if you do not wish to use the GPU, change the command to `\$ conda env create -f environment/environment_no_GPU.yml`.
3. Test your installation by running `\$ python3 test_HVQE.py`. This may take a minute.

You are all set for running a VQE for your own graphs and ansätze. The installation was tested on Debian, openSUSE, and macOS. Noisy simulations were only tested without GPU.