Institute for Theoretical Physics
University of Cologne

Prof. Dr. S. Trebst
B. Han, Q. Preiss, M. Yutushui

Quantum Computational Physics

Exercise Sheet 6

Summer Term 2026

Due date: Monday, 29.06.2026 **Discussion**: Tuesday, 30.06.2026

Website: thp.uni-koeln.de/trebst/Lectures/2026-QuantCompPhys.shtml

As mentioned in Tuesday's lecture, this final exercise sheet of the year will guide you through some of the calculations done in one the highlight papers reviewed in the lecture, namely 'Nishimori transition across the error threshold for constant-depth quantum circuits' which reports the realization of a Nishimori transition on one of IBM's 127-qubit quantum processors. We are going to redo the analysis of the paper using data – real quantum or artificial classical, and will you be able to tell? Let's get started!

Exercise 10: Can you find Nishimori's cat? 🔍

This exercise is about finding Nishimori's cat in the data we provide to you. But there is a tiny problem: Nishimori's cat is made of glass and we will have to look very carefully to be able to spot it! The only way we will be able to see it is by looking through our decoding lens.

Figure 1 – We (right) are searching for Nishimori's cat (left) which is invisible to the naked eye, since it is made of glass. Only by looking through our decoding lens, we will be able to spot it!

We provide data to you in this ILIAS folder as a compressed tar file. Download the file and save it into a new folder. Now you need to extract it using the following command:

bash

```
tar -xzf files.tar.gz
```

Once the files are extracted, you will find three code files (one .py and two .jl files) and a folder called `data`. The folder `data` contains the data we provide to you, which consists of "samples_bondstring..." files containing the measurement results of the auxiliary qubits, as well as "sitemeasurement..." files

containing the measurement results of the system qubits. You won't need to interact with these files manually, as we provide you with functions to load and manipulate the data.

You can load the data of the system qubits using the function

julia
```
loaddata(Ly::Int, tApi::Float64; kwargs...)
```

that we provide to you. The function takes the arguments `Ly` (the size in y-direction) and `tApi` (the coherent gate error t_A in units of π) and returns an array of size N_{qubits} x N_{samples} containing the elements ± 1. Specifically, we provide data for `Ly` $\in$ `[2, 3, 4]` (that corresponds to $N_{\text{qubits}} \in \{10, 28, 54\}$) and `tApi` $\in$ `0.0:0.0125:0.2375`. Setting some `kwargs` allows you to change the folders where the data is loaded from (which you should not need to do if you work in the provided folder structure) as well as some other parameters which we do not want to worry about at this point.

Let's start by looking at the magnetization $M = \sum_i \sigma_i$ (i.e. the sum of the measurement outcomes of all system qubits). The protocol is supposed to create a GHZ state for large enough t_A so we should see peaks at $M = \pm N_{\text{qubits}}$, right?

a) Plot the average magnetization of the data we provide to you for different values of t_A. What do you see? Can you spot Nishimori's cat (i.e. do you see the peaks at $M = \pm N_{\text{qubits}}$)?

Even though we may not have found the cat already, we are master detectives – so we won't give up that easily! We will now plot the "susceptibility" functions f and g for the data we provide to you. These functions are defined as

$$f := \frac{1}{N_{\text{qubits}}} \left(\langle M^2 \rangle - \langle M \rangle^2 \right), \tag{1}$$

$$g := \frac{1}{N_{\text{qubits}}^3} \left(\langle M^4 \rangle - \langle M^2 \rangle^2 \right). \tag{2}$$

b) Plot f and g as functions of t_A for the data we provide to you. What do you see? Can you spot Nishimori's cat now? Do your plots agree with the ones in the paper?

Oh no! We totally forgot that the cat (state) is glassy, so we are looking right through it. We will have to make use of our special decoding lens in order to make it visible!

In order to see Nishimori's cat we will have to decode the data. Decoding works essentially the same as on the previous exercise sheet, meaning that we have to match pairs of active stabilizers. This time though we will we be using the MWPM implementation provided by the package PyMatching (one of the state-of-the-art decoding libraries) to decode the data. In order to use it, you will have to install `pymatching`, as well as `numpy` and `scipy`. You install these packages by running:

bash
```
pip install pymatching numpy scipy
```

We recommend creating a new virtual environment for this – although you could also use the one you created for the Qiskit exercises.

As this exercise sheet is not supposed to be a tutorial for PyMatching, we will provide you with a small script that will do the decoding for you. To use this script, you will have to provide the size in y-direction `Ly` and the incoherent noise `meas_err`, which we will set to 0.0 for now. You can call the script by running

```bash
python decoder.py Ly meas_err
```

in the terminal (while you are in the directory where the script is located). This will create a folder `decoding` as well as files within it that contain the decoding results. We still have to apply these results to the data in order to make use of them. For that purpose we provide you with the Julia function `generate_correction_files(Ly::Int, tApi::Float64; kwargs...)`. Calling this function will generate the files that contain the decoded data.

c) Use the provided Python script to decode the data for `Ly` $\in$ `[2, 3, 4]` and `meas_err = 0.0`. After that run the Julia function `generate_correction_files` to apply the decoding results to the data.

Great, we unlocked a new item 🔍 – our decoding lens! Let's see if we can spot Nishimori's cat through it! You can load the decoded data using the `decoded` kword in the `loaddata` function, i.e.:

```julia
loaddata(Ly::Int, tApi::Float64; decoded=true, kwargs...)
```

d) Plot the average magnetization of the decoded data for different values of t_A. What do you see? Can you spot Nishimori's cat now? Also calculate the mean magnetization over all samples. Can you see anything in that mean?

e) Plot the functions f and g as functions of t_A for the decoded data. What do you see? Do your plots now agree with the ones in the paper?

Awesome! We found Nishimori's cat! What previously was just a – seemingly random – sequence of ± 1, now reveals to actually be a GHZ state! We can now see the peaks at $M = \pm N_{\text{qubits}}$ in the magnetization and the functions f and g show the expected behavior. All of this because we were able to decode the glassy GHZ state based on the measurement outcomes of the quantum computer.

But there is still one more step to go. Quantum computers are noisy, so let's see how incoherent noise affects the decoding of Nishimori's cat. Let's say we try to find Nishimori's cat for a fixed value of $t_A = 0.2\pi$ but we introduce additional incoherent noise (i.e. a vertical slice in the phase diagram below).

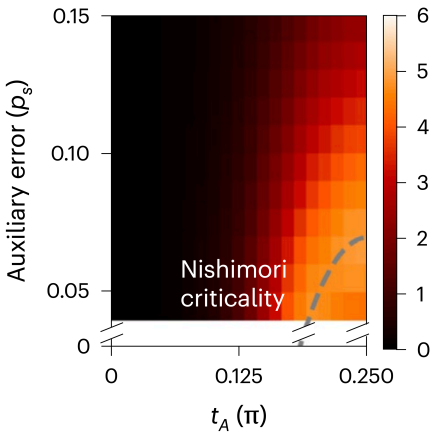

Figure 2 – Phase diagram from the paper. Plotted is the function g as a colormap for the decoded data as a function of the coherent gate error t_A and the incoherent noise p_s (also called `meas_err`) for the largest system size $N_{\text{qubits}} = 54$ (also referred to by `Ly = 4`).

We provide data to you for different values of incoherent noise `meas_err` $\in$ `0.0:0.005:0.15`. You can use the `loaddata` function, as well as the `generate_correction_files` function with the noisy data by setting the `meas_err` kword to the desired value. In order to decode the noisy data, you will have to run the Python script with the corresponding `meas_err` value by, e.g. using a small bash for loop.

bash

```
for meas_err in $(seq 0.0 0.005 0.15)
do
    python decoder.py Ly $meas_err
done
```

f) Plot f and g (for the decoded data) as a function of the incoherent noise for $t_A = 0.2\pi$. What do you observe?

g) ($\star$) We provide data for the whole phase diagram in the paper (for `Ly = 4`). Can you reproduce it? That is, plot g as a colormap for the decoded data as a function of the coherent gate error t_A and the incoherent noise p_s.