## Contents

## 1 Introduction

This document is intended to register the activities I have been working on since the end of my Master Thesis. I will be writing different sections and introducing the topics I will work on. Most of them will be unfinished due to time issues, but I hope this guide can be useful for anyone who wants to carry on with the present studies.

## 2 Variational Quantum Singular Value Decomposition

Alba had the idea to combine somehow the randomized protocol into a sort of VQSVD. I try to implement some ideas and run some tests, but there is nothing I am proud enough to share here. The most of it is useless, as it leads to nothing.

## 3 Estimation of $\mathrm{Tr}\{\rho^3\}$ and the implementation of the PPT-criterion

As starters, I recommend the Appendix of my Master Thesis. There you can find an explanation of how to estimate $R_3 = \mathrm{Tr}\{\rho^3\}$ and use that as an entanglement checker throughout the PPT-criterion. The expression we are using for the $R_3$ estimation is

$$\mathrm{Tr}\left\{\rho^3\right\} = \frac{1}{2}\left((\mathcal{D}+1)(\mathcal{D}+2)\sum_s \overline{P_U(s)^3} - 3\,\mathrm{Tr}\left\{\rho^2\right\} - 1\right). \tag{1}$$

It can be straightforward derived from my Master Thesis (you can also find some references there in case you are curious). It is worth-mentioning that grater orders $R_n$ can be derived with ease in a recursively way, but requires to solve an equation with some constraints that I do not find straightforward to generalize. For this reason I decided to focus exclusively on $R_3$, as it seems the most useful value at the moment. We must also say that Eq.1 has been derived considering the exclusive usage of the Global approach. For the Local approach future research must be performed. There is this paper [https://arxiv.org/abs/2007.06305] where local measures are treated, but I do not understand how the randomized measures are used on the implementation of the protocol. Further research must be conducted here before I leave !!!!.

Another major issue I found during the implementation of the code is the probability cubed $P_U(s)^3$ estimator. It is important to find an unbiased estimator, otherwise the algorithm's performance will not increase with the number of measurements ($N_M$) during one of the circuit's execution. In this paper, [https://doi.org/10.1103/PhysRevA.97.023604], you can find some criteria to define the probability squared estimator $P_U(s)^2$, which I used on the original randomized protocol for the $R_2$ estimation. In this paper, they stated that "*a unique faithful estimator $\tilde{P}_n$ [for $P^n$] is constructed using an ansatz $\tilde{P}_n = \sum_{k=0}^{n} \alpha_k \tilde{P}^k$ [where $\tilde{P}$ is the well-known estimator for the probability] and the defining condition $\mathbb{E}[\tilde{P}_n] = P^n$* " . Using these criteria, and with the help of algebra and time, one can finally arrive to

$$\tilde{P}_3 = \frac{2\tilde{P} - 3N_M\tilde{P}^2 + N_M^2\tilde{P}^3}{(N_M-1)(N_M-2)}. \tag{2}$$

This estimator has been proven to be unbiased and it has been correctly implemented on the final code, with great results. Down below you can check at a rough example for the $R_3$ estimation of the GHZ state. However, $R_3$ estimation will be added to the code on the repository. I should show how is easily calculated with myfunctions !!!!!

```python
import numpy as np
from myfunctions import Randomized_Renyi_2 as random_renyi2     #function for the randomized protocol.
from mycircuits import GHZ

nqubits = 6     #number of qubits
c = GHZ(nqubits)        #circuit execution for the GHZ state
copy = c.copy()

matrix = copy.unitary()     #quantum state as a matrix. Useful math operations

f = random_renyi2()
N_A = [0,1,2]
N_U = 1000
N_M = 256

f.Global(c, N_A, N_U, N_M)      #execution of the Global random protocol

entropy = f.entropy()
purity = 2**(-entropy)
freq = f.frequencies()

lista_p = []
bits = all_binary_iterations(3)
for i in range(len(freq)):
    iter_u = 0
    for j in bits:
        p = freq[i][str(j)]/N_M
        iter_u += (2*p - 3*N_M*p**2 + N_M**2*p**3)/((N_M - 1)*(N_M - 2))

    lista_p.append(iter_u)

mean = np.mean(lista_p)

p3 = 0.5*((2**3 + 1)*(2**3 + 2)*mean - 3*purity - 1)

initial_state = np.zeros(shape = (2**nqubits, 2**nqubits))
initial_state[0,0] = 1
final_state = matrix @ initial_state @ matrix.T.conj()
#print(final_state)

n1, n2 = int(len(N_A)), int(nqubits-len(N_A))       #est  hecho para partir A y B por la mitad justo. De otra manera ni idea.
reshaped_psi = np.reshape(final_state, [2**n1, 2**n2, 2**n1, 2**n2])
rho_A = np.trace(reshaped_psi, axis1=1, axis2=3)
#print(rho_A)

U, D, V_dagger = np.linalg.svd(rho_A, full_matrices=True)

#print(D)

puri = 0
cubbed = 0
for i in range(len(D)):
    puri += D[i]**2
    cubbed += D[i]**3

print('Real: ', puri, 'Estimated: ', purity, 'Diff: ', np.abs(puri - purity))
print('Real: ', cubbed, 'Estimated: ', p3, 'Diff: ', np.abs(cubbed - p3))
```

```

#It should print something like this

Real:  0.4999999999999998 Estimated:  0.5016240808823529 Diff:
    0.0016240808823531072
Real:  0.24999999999999983 Estimated:  0.2573561508510884 Diff:
    0.007356150851088594
```

Listing 1: Rough implementation of the $R_3$ estimation.

We have now a method to compute $R_3$ without too much problems. Thus, let's move on to the next step: estimate entanglement through the PPT-condition. I recommend one again to read my Master Thesis Appendix, but basically, given this statement

$$\rho_{AB} \in PPT \implies p_3 \geq p_2^2, \qquad \text{with} \qquad p_n = \mathrm{Tr}\left\{(\rho_{AB}^{T_A})^n\right\} \quad \text{for} \quad n = 1, 2, 3\ldots \tag{3}$$

we name the $p_3$-PPT condition to the contra-positive of the previous assertion. Thus, if $p_3 < p_2^2$, then $\rho_{AB}$ violates the PPT condition and $A$ and $B$ must be entangled. As we can see, we now need to estimate $p_n$, which is a similar magnitude to $R_n$, with the difference that we should transpose one of the two subspaces (for example subspace A) of the original quantum state.

During this the implementation of $p_n$ estimation I encounter some problems. One of them was: how do I apply a transposition of just one subspace during the quantum circuit execution? I could not find an answer to that (I should keep searching !!!!) so I just come up with something not as good as I would like. I decided to initialize the state already at the A-transposed (remember that using qibo we can always initialize the quantum state at some arbitrary $\rho$), just to check that everything worked fine and until I found a better solution.

First, I programmed a function inside the initial class, called *f.entanglement_checker()*, that must be used after running the Global protocol (which allows to get the data). Basically, what *f.entanglement_checker()* does is compute $R_3$ and compare it with $R_2$, without transposing (as I do not know how to do it yet). Code 2 shows how the implementation on my function could work.

```
#remember that only works for global
nqubits = 6
c = GHZ(nqubits)

f = random_renyi2()
N_A = [0,1,2]
N_U = 1000
N_M = 256

f.Global(c, N_A, N_U, N_M)
f.entanglement_checker()
#should print something like

p3 value: 0.24955823732051918
p2^2 value: 0.24990212355226290

State is entangled
```

Listing 2: Implementation of the entanglement checker that compares $p_3 < p_2^2$ if the initial quantum state has already been transposed

However, I found that due to numerical values, it could be better to add some kind of window for the values to be accepted. For example, for the GHZ state, which is pure, we should expect to find the same value for $p3$ and $p_2^2$. Thus, equality would hold and we should not be able to tell if the state is entangled using the PPT-condition (although for obvious reasons it is entangled and can be demonstrated if it gives a value when computing the Rényi entropy. Also we may investigate more to see if that is due to being pure or maximally entangled or both).

Then, I run some tests on the 2-qubits Werner state, as it is common and well studied. Although we can write it as

$$W_{AB} = \lambda \left|\Psi^-\middle\rangle\middle\langle\Psi^-\right| + \frac{1-\lambda}{4} I_{AB}, \qquad |\Psi^-\rangle = \frac{|01\rangle - |10\rangle}{\sqrt{2}} \tag{4}$$

with $\lambda \in [-1/3, 1]$, entangled fort $\lambda > 1/3$, I decided to write the Werner state using explicitly the matrix form,

$$W_{AB} = \begin{pmatrix} \frac{p}{3} & 0 & 0 & 0 \\ 0 & \frac{3-2p}{6} & \frac{-3+4p}{6} & 0 \\ 0 & \frac{-3+4p}{6} & \frac{3-2p}{6} & 0 \\ 0 & 0 & 0 & \frac{p}{3} \end{pmatrix}. \tag{5}$$

In this case, Werner state are separable (thus not entangled) for $p \geq 1/2$, and entangled otherwise. I searched for a way to compute a Werner state in a quantum circuit using real quantum gates, but I did not find something I could work with. I found this article [https://arxiv.org/abs/1912.06105] that showed how to compute the Werner state. Problems are: (1) you need more than 2 qubits, aka auxiliary qubits in the circuit are needed, which makes unfeasible the application of the randomized protocol, at least the way I have it programmed (Maybe I can add entropies somehow and estimate the value or something); (2) there was a circuit with only the 2 main qubits of the WS, but I could not be programmed on Qibo, as we need a specialized circuit (I tried to use ifs and elses but I does not work, I do not think I understand the concept of specialized).

This way, I finally computed $p2$ and $p3$ by initialising the state in $W_{AB}^{T_A}$ and represent the values as a function of the value $p$ from Eq.5. Results are shown at Figure 1 (Re do it with $p^2$ and also with good axis). As we can see, the estimated value starts to diverge from the real one as entanglement in the system is presented (remember the state is entangled for $p < 1/2$). It is also worth-mentioning that the error is bigger during the estimation of $p_2$ than in $p_3$. That may be because the errors, which are less than 1, decreases with the exponential power. We should one again have a look to the appendix of [https://arxiv.org/abs/2007.06305]

This results seemed odd, so I decided to test it again on the WS, now by computing $R_n$, just to see if the randomized protocol encounter a great limitation with this concrete state (and also to check the effect of entanglement on the protocol). Figure 2 shows the results.

As expected, the randomized protocol still works perfectly fine, but we can also appreciate how error arises as son as entanglement is presented in the system, although not as intense as in the previous one, where the transposition was conducted.

In order to test if the problem during the $W_{AB}^{T_A}$ estimation was caused by errors raised during the randomized nature of the protocol, Alba recommend me to test it against the Clifford group protocol. However, I cannot implement it to estimate $p_3$, as remember that I only have the Clifford group protocol for the Local approach (as implement a global approach may be inefficient due to the enormous amount of components of the group). In order to do that, I had to create a new function to apply the Clifford protocol to an initial

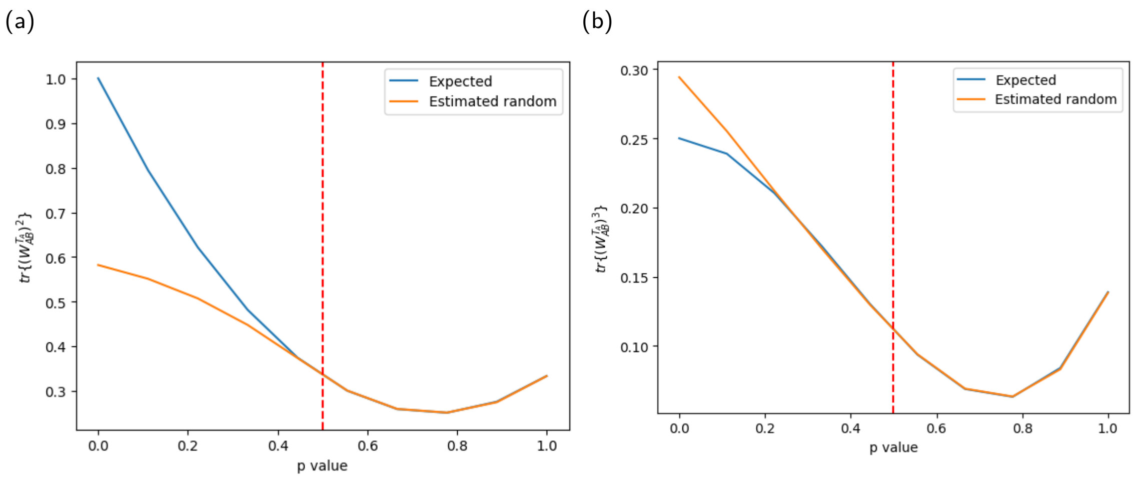

Figure 1: Trace value (Y-axis) as a function of the value $p$ (X- axis) given by the randomized protocol (estimated) and compared to the real one (expected), for the $W_{AB}^{T_A}$ state.

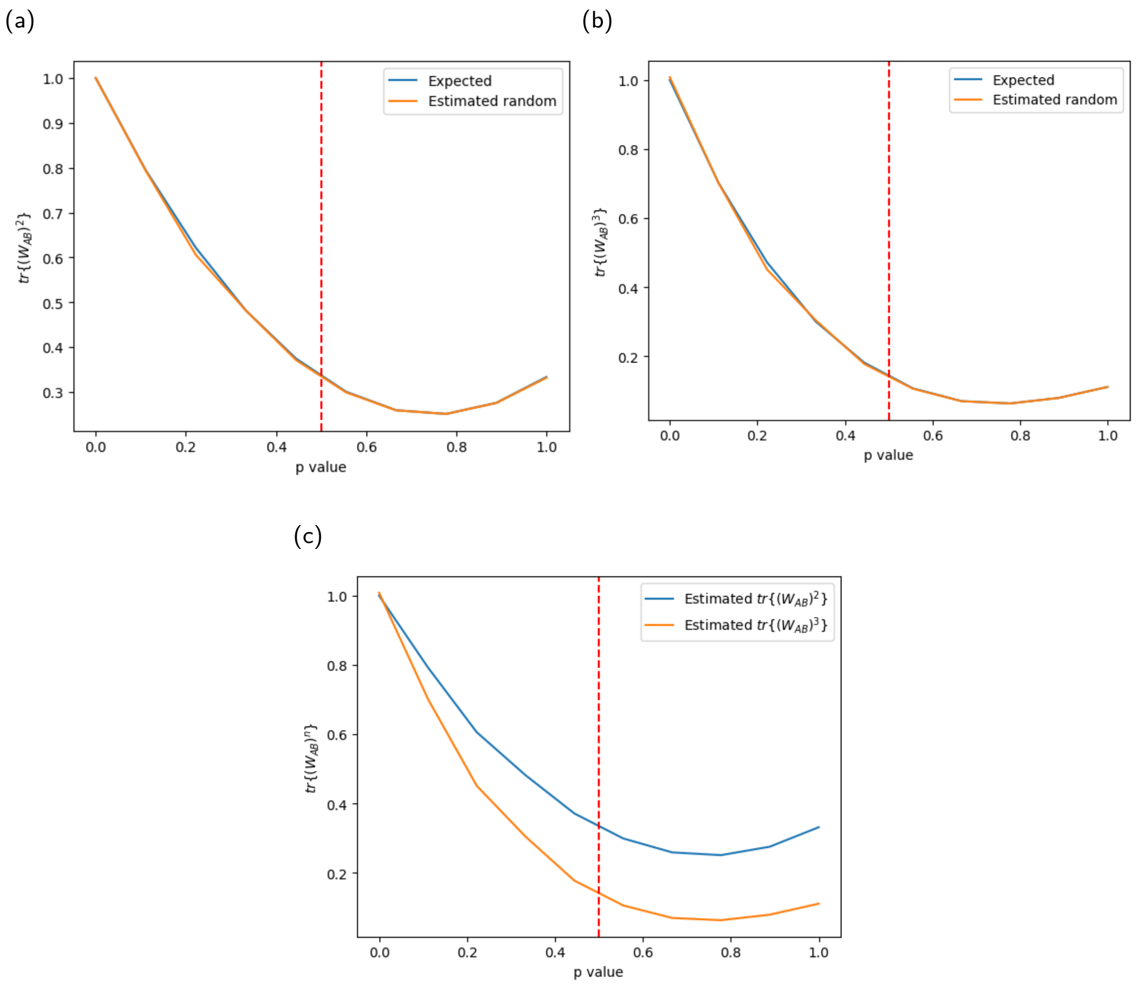

Figure 2: Trace value (Y-axis) as a function of the value $p$ (X-axis) given by the randomized protocol (estimated) and compared to the real one (expected) for the $W_{AB}$ state.

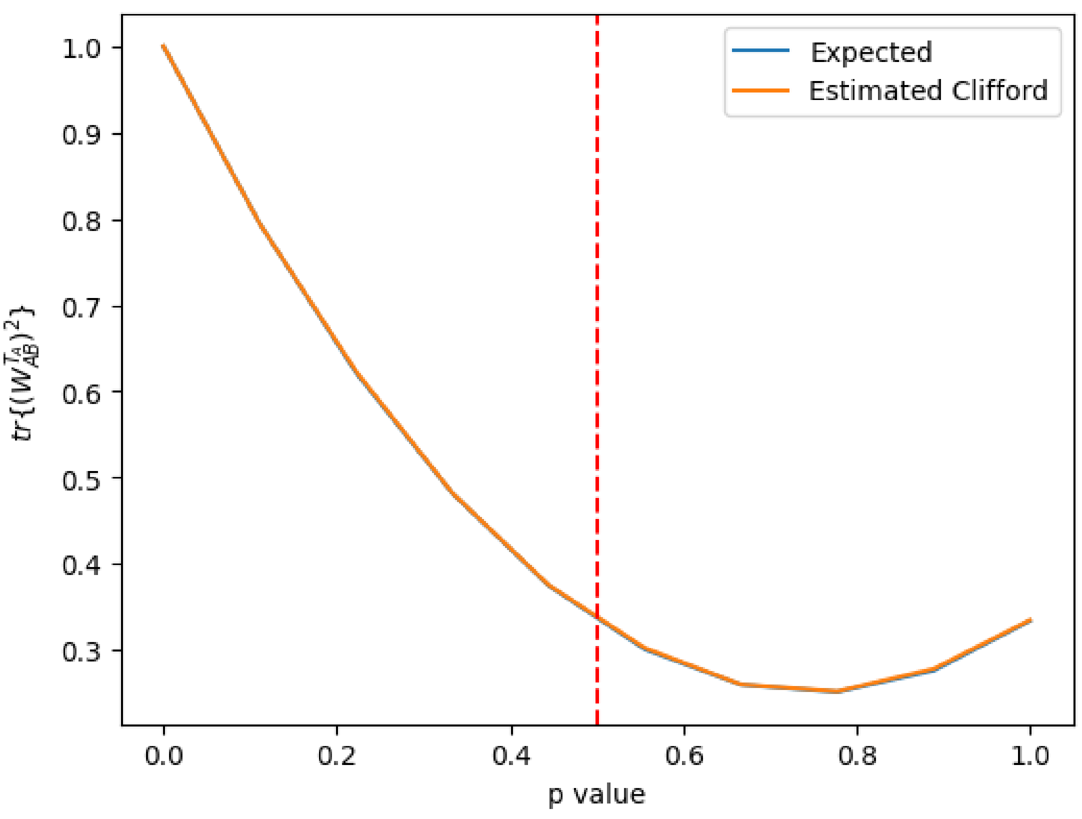

Figure 3: Trace value as a function of the parameter $p$ given by the Clifford protocol (estimated) and compared to the real one (expected) for the $W_{AB}^{T_A}$ state.

matrix state. After that, I test it without problems. Figure 3 shows the results, which are perfectly what we expected, with the Clifford approach giving the exact value.

It is also important to notice that the expected $p2$ and $R_2$ (aka the trace value for the $W_{AB}^{T_A}$ and $W_{AB}$) are the same. It would be nice to prove theoretically why and add some nice figures.

## 4 Meta-VQE assistance

Berta asked me to have a look at some of their code for the Meta-VQE project, because some of the results regarding the randomized protocol did not fit well. After running some tests (like the Clifford approach) I found out that Berta was mistakenly comparing the Von Neumann entropy, provided by the callback of Qibo, with the 2-order Rényi entropy given by the randomized protocol. Of course, although both measure give insight of the entanglement nature of the quantum state, they do not math the exact value (except when being maximally entangled). Thus, there was no way to compare the results given by the randomized protocol with the exact values. For this reason, I decided to enter into the Qibo repository and have a look at the entanglement entropy measure based on the callback.

Basically, I took the existent class *EntanglementEntropy(Callback)*, which can be found at *callbacks.py* and make a copy with the name of *RenyiNEntropy(Callback)*. Inside this class, I made some modifications in *apply()* and *apply_density_matrix()* functions, changing the *backend.entanglement_entropy()* function into *backend.renyin_entropy()*. This last function will be defined somewhere else. I added Code 3 inside the *abstract.py*, just in case. Any function not defined in this file will be redirected to *numpy.py*. Here is where the true function must be defined. One again, I took the *entanglement_ entropy()* function as a base, and I implemented here the final function of *renyin_ entropy()*, Code 4.

```python
@abc.abstractmethod
    def renyin_entropy(self, order, rho):  # pragma: no cover
        """Calculate renyi entropy of order 2 of a reduced density matrix.
    """
        raise_error(NotImplementedError)
```

Listing 3: Abstract.py

```python
def renyin_entropy(self, order, rho):
        from qibo.config import EIGVAL_CUTOFF

        # Diagonalize
        eigvals = self.np.linalg.eigvalsh(rho).real
        # Treating zero and negative eigenvalues
        squared_eigvals = eigvals[eigvals > EIGVAL_CUTOFF]**order
        spectrum = (1/(1 - order)) * self.np.log(squared_eigvals)
        summer = self.np.sum(squared_eigvals)
        entropy = (1/(1 - order)) * self.np.log2(summer)
        return entropy, spectrum
```

Listing 4: numpy.py

With all these changes, it is possible to run this "new" entanglement measure entropy. The procedure to follow is the same as for the Von Neumann entropy, represented on Code 5.

```python
from qibo import models, gates, callbacks
# create entropy callback where qubit 0 is the first subsystem
entropy = callbacks.RenyiNEntropy([0], compute_spectrum=True)
# initialize circuit with 2 qubits and add gates
c = models.Circuit(2)
# add callback gates between normal gates
c.add(gates.CallbackGate(entropy))
c.add(gates.H(0))
c.add(gates.CallbackGate(entropy))
c.add(gates.CNOT(0, 1))
c.add(gates.CallbackGate(entropy))
# execute the circuit
final_state = c()
print(entropy[:])
# Should print [0, 0, 1] which is the entanglement entropy
# after every gate in the calculation.
print(entropy.spectrum)
# Print the entanglement spectrum.
```

Listing 5: Example for Rényi entropy.

Figure 4 shows the result given by the notebook provided by Berta, where this new callback has been introduced and the randomized protocol for the 2-order Rényi entropy has been implemented with $N_U = 1000$. It is interesting to see the same problems we have been facing before. The values given by the protocol differ more from the expected behaviour as the entanglement (and thus entropy value) increases. Any idea why???.

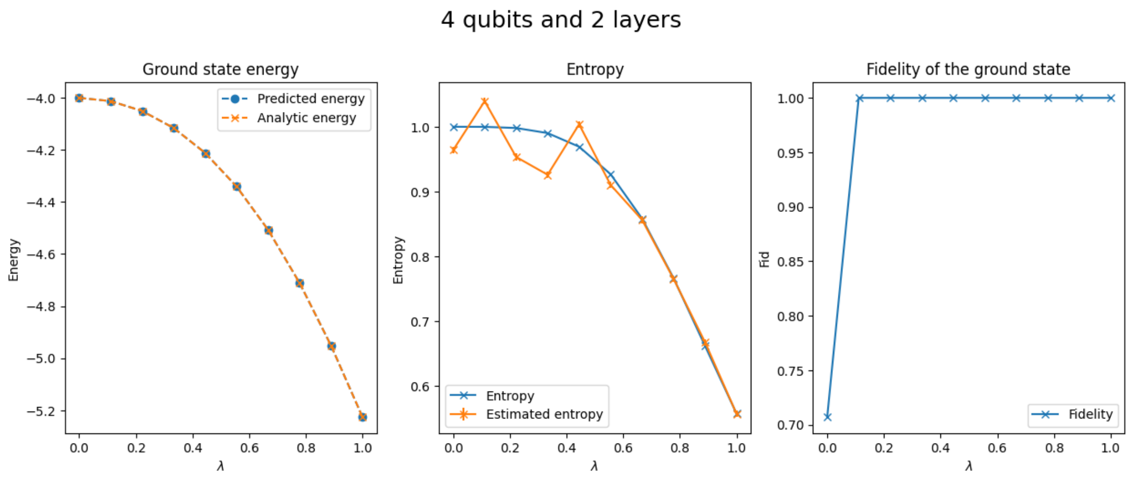

Figure 4: Result provided by Berta's notebook execution.

## 5 Qiskit implementation and IBM execution

The general idea is to translate the randomize protocol into the Qiskit package and execute it in the IBM quantum hardware.

First of all, I started reading the Qiskit documentation to refresh my memory and relearn the basics. I am still having problem to understand the ClassicRegister during a circuit execution, but I think I have it under control.

Then, when I had things clear, I began to translate the code. I must say it was simple, as it is really similar to Qibo. I ran it on the Aer simulator, to test that everithing was fine, and it worked perfectly. Thus, next step was to implement the code to run the job on a real device. The issue I found was that I cannot send multiple jobs (as I was doing during the simulations), because the ques and waiting time is far too long. For this reason, I reprogrammed once again the code to prepare a number of $N_U$ different circuits and send them at one as an only job. Strangely, that increases the execution time during my test on the Aer simulator. However, I test it on the real hardware and it worked fine. I have been using *QiskitRuntimeService*, which returns the probabilities instead of the counts. However, for some reasons I had problems when retrieving them during the execution on the notebook (I got some post-processing errors). So, I took the rough data from the IBM Quantum Platform and treat the data my self. I tested on the GHZ state and found an entropy value of $S_2 = (1.2438 + -0.0097)$. As you can see, it diverges from the exact 1.0 value we expected, and that is probsabliy because I send the job to *ibmq_belem*, which does not have the qubit interconnectivity I expceted. For this reason, I must repeat the procedure in another quantum hardware, making sure that the CNOT gates are applied in qubits with existent connectivity.

I decided now to send the job to *ibm_perth* (Figure 5 shows how is the inter-connectivity between the qubits) and create a GHZ state using only qubits 0, 1, 3, 5 and 6. I executed the randomized measure over subsystem $N_A = [0, 1, 3]$. The obtained result was now of $S_2 = (0.9758 \pm 0.0083)$, a much better approximation to the real value. However, and for some reason I cannot understand, some of the frequencies results were negative values, so I treated them in absolute values (as negative ones raised some errors). Multiple tests should be performed in order to proof the reliability of the randomized protocol, but due to time constraints I did not performed them.

I must also mention that the Qiskit implementation of the randomized protocol has

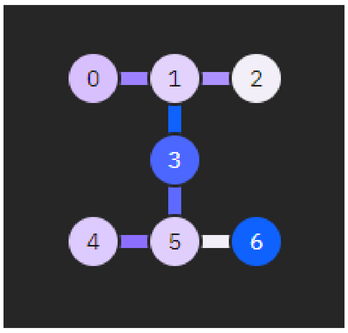

Figure 5: IBM perth quantum computer's qubit distribution.

only been performed for the Local approach, as the Global may not be feasible for the real hardware (May be I should try it nevertheless.)

## 6 Tequila implementation

The idea is to translate the randomized protocol again, from qibo into tequila. That is because the Meta-VQE they are programming seems to work better on tequila (I do not know the technical details). So, Alba asked me to translate it.

First, I read all the tutorial presented by tequila on Github, in order to get familiar with the package. Meanwhile, I also took a look into the main code of tequila, so I could understand better how everything worked. When I finally though I understand enough, I started programming the basics.

First, I created a function that returns a circuit with the GHZ state, so I could use it to run some tests. Then, I realized that after a circuit execution, the output was a dictionary containing a number (that is associated with the output bitstring) and its count. As I need the specific bitstring output for the randomized algorithm, I searched in tequila's repository for a way to obtain them straightaway. Unfortunately, I did not find it (which I still think it is strange, so probably I was not able to find it, but believe me, I spend too much time), so I created a function that given a number, returns the associated bitstring with the right amount of length.

Next step, I needed a way to perform measurements in random basis. I used one again qibo to produce random unitary matrices following the distribution given by the Haar measure (although qiskit also has that ability. In fact, it is really easy to program a function by ourselves that returns random unitaries; I recommend to take a look into the qibo repository). However, I could not find a way to apply a unitary gate coming from a unitary matrix into a qubit (qibo and qiskit both present better and easier implementations), so I used a function from qiskit that allowed me to decompose any unitary gate using any kind of basis decomposition. Thus, I choose the U3 basis to decompose the random matrix, took the angles that defined it and apply a U3 gate (which is perfectly define on tequila) after the qubit I want to measure. In principle, if I take one random unitary, I decompose it and I form a U3 gate, the final quantum state is not strictly the same, they differ in an absolute phase. Fortunately, probability outcomes do remain still the same, which is what we need to initialize the randomized protocol.

Finally, with all that I had everything I needed to translate the code. In this case, I translated the Local, the Global and the Clifford approaches. I find a little problem during the Global implementation, as I had to change its core more than I though. But, after some tests, everything seems to work fine.

Nevertheless, Berta recently told me she had a problem during the execution of the protocol. It seems she encounter an issue when deafening the variables of the quantum gates. I must say this problem is on me, as I did not conducted any tests regarding the definition of variables during the circuit execution. So, I revisited the documentation and my code, and I fixed it. I just added an optional variable, which I just called *variables*, which is a dictionary that much contain the variable with its associated values.