Implementing the Quantum Fourier Transform and Phase Estimation Algorithm

, Markus Fuhrmann [1]
[1] Department of Physics, University of Oslo, Norway

The topic of this report are complex quantum algorithms, namely the Quantum Fourier Transform (QFT) and the Phase Estimation (PE). The theory and of the QFT and the PE as building blocks for more complex algorithms are discussed. Their implementation happens without the use of any quantum computing libraries. Both algorithms have been shown to work properly to the expected extent, with the exception of the accuracy of the PE when predicting random eigenvalues.

***Index Terms*—QFT, Phase Estimation, Quantum Computing, Quantum Algorithms.**

I. Introduction

THE Quantum Fourier Transform (QFT) is a quantum algorithm that performs the Discrete Fourier Transform (DFT) on a given 2^n dimensional vector, where n is the number of qubits the algorithm is running on. It is one of the fundamental building blocks of more complex quantum algorithms like the phase estimation or Shor's algorithm. The phase estimation algorithm is used in finding the eigenvalues of unitary operators and Shor's algorithm uses the reduced complexity of the QFT ($O(n \log n)$ compared to the DFT ($O(n^2)$) to speed up integer factorization.
In the first chapter we will recapitulate the theory behind the QFT and explicitely calculate some examples. After implementing the QFT circuit from scratch, we will compare it to Qiskit's QFT function. In the second chapter we will have a look at the phase estimation algorithm and follow the same procedure as for the QFT.

II. Basic Principles

A. Fourier Transformations

In 1822 Fourier had the idea, that any function can be expressed by a linear combination of sines (or an infinite series). This lead to one of the most important mathematical tools in signal processing, the Fourier Transform (FT). Additionally it has many other applications, like we will see in this project. To put it in words, the Fourier Transform of a function decomposes it into frequencies appearing in the function and it's amplitude. Let's look at an example to understand what this means. The FT of a function can be calculated via

$$F\{f(t)\} = f(\zeta) = \int_{-\infty}^{\infty} \mathrm{d}t f(t) \exp\{-i2\pi\zeta t\}\,.$$

In fig. 1 a signal has been generated using the FTExamples.ipynb. After putting in some frequencies and amplitudes, it creates a single wave packet by overlapping sinusoidal terms with given parameters. Using numpy, the FT of that signal is calculated, and shows exactly what frequencies and amplitudes the original signal contains. When the function, that is getting transformed, is not continuous, but discrete, the integral collapses to a summation, leaving us with the Discrete Fourier Transform (DFT). It maps a sequence of complex numbers into another sequence of complex numbers, which is less visual, than what we had earlier with the frequencies and amplitudes. However, thinking about continuous functions being discrete, but in the limit of infinitely small distances between adjacent points, it still does the same thing [1]. The important difference is, that the DFT can be applied to non-continous functions, because it can handle discrete jumps. Mathematically the DFT is defined as

$$X_{\mathrm{k}} = \frac{1}{\sqrt{N}} \sum_{\mathrm{n}=0}^{\mathrm{N}-1} x_{\mathrm{n}} \exp\left\{-i2\pi \frac{k}{N} n\right\} \tag{1}$$

with its inverse transformation

$$x_{\mathrm{n}} = \frac{1}{\sqrt{N}} \sum_{\mathrm{l}=0}^{\mathrm{N}-1} X_{\mathrm{k}} \exp\left\{i2\pi \frac{k}{N} n\right\}$$

A small example hinting at its usability and possible implementation in a quantum circuit is following: Assume we want to transform the vectors $|0\rangle$ and $|1\rangle$. We then get

$$\begin{pmatrix}1\\0\end{pmatrix} = \begin{pmatrix}X_0\\X_1\end{pmatrix} \quad = \begin{pmatrix}1+0\cdot e^{-i2\pi\cdot 0/2}\\1+0\cdot e^{-i2\pi\cdot 1/2}\end{pmatrix} = \begin{pmatrix}1\\1\end{pmatrix}$$
$$\begin{pmatrix}0\\1\end{pmatrix} = \begin{pmatrix}X_0\\X_1\end{pmatrix} \quad = \begin{pmatrix}0+1\cdot e^{-i2\pi\cdot 0/2}\\0+1\cdot e^{-i2\pi\cdot 1/2}\end{pmatrix} = \begin{pmatrix}1\\-1\end{pmatrix}$$

We immediately see, that in this case, applying the DFT to our states, gives the same result as simply applying the Hadamard gate to them, motivating us to implement the DFT in a quantum circuit!

[1] A website, that visualizes the DFT can be found here: "https://madebyevan.com/dft/"

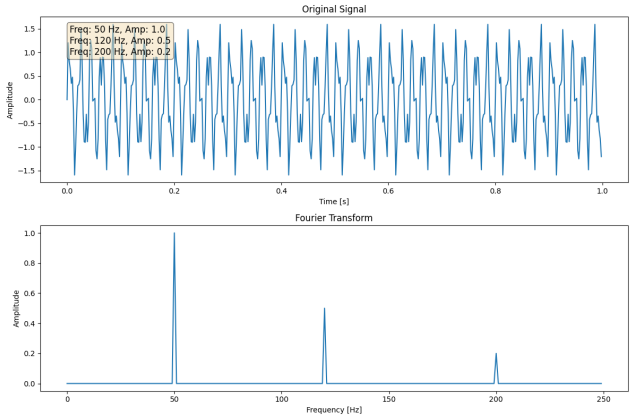

Figure 1: Example of a signal created by overlapping sinusoidal functions with different frequencies and its FT, returning which frequencies and amplitudes are contained.

B. Phase Kick Circuit

Before we dive deeper into Quantum Fourier Transform (QFT), we have a closer look at the phase kick circuit and the binary encoding, which help us understand the QFT.
The phase kick circuit is one of the elemental building parts, as it allows us to add phases in a controlled way to the state of a specific qubit. A simplified version of it can be seen in figure 2. Let's calculate the states after each gate. After the Hadamard gate the state reads

$$|\psi_1\rangle = |+\rangle \otimes |1\rangle = \frac{1}{\sqrt{2}} \left(|0\rangle |1\rangle + |1\rangle |1\rangle\right)$$

and after the Controlled-S operation, the final state becomes

$$\begin{aligned} |\psi_2\rangle &= \frac{1}{\sqrt{2}} \left(|0\rangle |1\rangle + |1\rangle S |0\rangle\right) \\ &= \frac{1}{\sqrt{2}} \left(|0\rangle |1\rangle + |1\rangle e^{i\pi/2} |0\rangle\right) \\ &= \frac{1}{\sqrt{2}} \left(|0\rangle + e^{i\pi/2} |0\rangle\right) |1\rangle \end{aligned}$$

We see, that the state of the second qubit stays unchanged and that we *kicked* the phase from qubit 2 (because we are applying the S-Gate to it) to qubit 1, the controlling qubit.

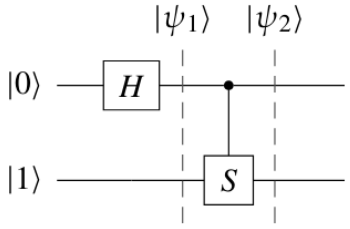

Figure 2: Simpflified Phase Kick circuit [1]

III. QFT

After discussing some elementary prerequisites regarding quantum circuits and FT, we can now start on a quantum algorithm to implement the DFT on a quantum computer.

A. QFT - Theory

Consider a n qubit system with basis vectors $|j\rangle$ now. The DFT for one of the basis states reads

$$\mathrm{DFT}\,|j\rangle = \frac{1}{\sqrt{N}} \sum_{\mathrm{k}=0}^{\mathrm{N}-1} \exp\{2\pi i j k/N\}\,|k\rangle$$

and for arbitrary states

$$\mathrm{DFT}\,|\psi\rangle = \sum_{\mathrm{j}=0}^{\mathrm{N}-1} x_{\mathrm{j}}\,|j\rangle = \sum_{\mathrm{k}=0}^{\mathrm{N}-1} y_{\mathrm{k}}\,|k\rangle$$

with

$$y_{\mathrm{k}} = \sum_{\mathrm{a}=0}^{\mathrm{N}-1} \exp\{2\pi i k a/N\} x_{\mathrm{j}}$$

being the DFT of the amplitude x_{j}.
We can derive a unitary matrix representing the transformation from this. Define $\omega_{\mathrm{N}} = 2\pi i/N$, then the unitary matrix becomes

$$U_{\mathrm{DFT}} = \frac{1}{\sqrt{N}} \sum_{\mathrm{x}=0} 1\mathrm{N} - 1 \sum_{\mathrm{y}=0}^{\mathrm{N}-1} \omega_{\mathrm{N}}^{-\mathrm{xy}}\,|x\rangle\langle y|$$

which looks written out like this:

$$U_{\mathrm{DFT}} = \frac{1}{\sqrt{N}} \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \omega^3 & \cdots & \omega^{N-1} \\ 1 & \omega^2 & \omega^4 & \omega^6 & \cdots & \omega^{2(N-1)} \\ 1 & \omega^3 & \omega^6 & \omega^9 & \cdots & \omega^{3(N-1)} \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ 1 & \omega^{N-1} & \omega^{2(N-1)} & \omega^{3(N-1)} & \cdots & \omega^{(N-1)(N-1)} \end{bmatrix}$$

Each state has a binary representation, indicating the state of each qubit

$$j = j_1 j_2 j_3 ... j_{\mathrm{n}}\,.$$

With this we can introduce a short hand notation for the exponent, namely

$$0.j_1 j_2 ... j_{\mathrm{n}} = \frac{j_1}{2} + \frac{j_2}{2^2} + ... + \frac{j_{\mathrm{n}}}{2}$$

Writing then the basis state in binary representation as the tensor product of all qubit states, and applying the QFT to that expression, we get

$$\mathrm{DFT}\,|j_1 j_2 ... j_{\mathrm{n}}\rangle = \frac{1}{\sqrt{N}} \sum_{\mathrm{k_b}\in 0,1} \exp\left\{2\pi i j/N \sum_{\mathrm{a}=0}^{\mathrm{n}-1} j_{\mathrm{a}} 2^{\mathrm{n}-\mathrm{a}}\right\} |k_{\mathrm{b}}\rangle \tag{2}$$

$$= \frac{1}{\sqrt{N}} \bigotimes_{\mathrm{k}=0}^{\mathrm{n}-1} \left[|0\rangle + \exp\left\{2\pi i j 2^{k-n}\right\} |1\rangle\right] \tag{3}$$

Expressing the DFT operator through this tensor product makes it easier for us to derive a circuit. As seen in **??**, the DFT

operator for one qubit is simply the Hadamard gate. This seems like a good place to start and after applying the hadamard gate to the first qubit our transformed state becomes

$$\text{DFT}\,|j\rangle = \frac{1}{\sqrt{2}}\left[|0\rangle + \exp\{2\pi 0.j_1\}\,|1\rangle\right] \otimes |j_2 ... j_\text{n}\rangle]$$

Comparing this to 3 we notice, that the phase, does not seem to be quite the same, as we would expect. To fix this we need to perform some rotations on that specific qubit with the rotation gates

$$R_\text{k} = \begin{pmatrix} 1 & 0 \\ 0 & \exp\{\frac{-2\pi i}{2^k}\} \end{pmatrix}$$

If we apply the gates R_2, R_3 and so on, the phase changes step by step to what we need and our transformed state becomes

$$\text{DFT}\,|j\rangle = \frac{1}{\sqrt{2}}\left[|0\rangle + \exp\{2\pi 0.j_1 j_2 ... j_\text{n}\}\,|1\rangle\right] \otimes |j_2 ... j_\text{n}\rangle] \tag{4}$$

This procedure will then be repeated for the next qubit and so on until the final qubit, which only gets a hadamard and no phase gates. This circuit is what we understand by QFT, because it can be executed on a quantum computer. A schematic of this circuit for 3 qubits is shown in fig. 3. There are a few things to notice about this circuit. First, notice the phase kick circuit as building block of the QFT circuit. Besides that, the schematic in fig. 3 also shows a SWAP gate in the end. This swap operation, is necessary to turn the qubits in the correct order with respect to the DFT definition. I don't quite understand where and when this becomes important in the derivation, but it does at some point. Later, when I talk about the implementation of the QFT I will mention the SWAP gates again.

Whether to choose QFT or the DFT for a problem depends on a few aspects. First and foremost should be the point, that the QFT is faster than the DFT, which would make it the first choice in any case. However, when regarding accuracy, the QFT is fighting against time, whereas the DFTs accuracy doesn't change over time. Decoherence, noise and gate imperfections make it hard for QFT algorithms to run for a long time. Another thing I have come across, is the amount of storage space needed, although I don't quite understand it. For example when I tried to calculate the QFT for 20 qubits, my VS Code told me it had run out of memory. This probably is the case, because of the vectors and matrices the program has to store, for 20 qubits the state vectors have dimensionality $2^{20} = 1048576$ and the operation matrices have the sqaured amount. This definitely is an issue for the DFT, when regarding scalability. For the QFT I am a bit unsure, whether this is constraining its applicability or not, since you would normally only store the name of the gate operations, but not their mathematical expression. And you only send a task/series of gate expressions to the quantum computer? I don't know enough about quantum computer architecture though to really judge about this.

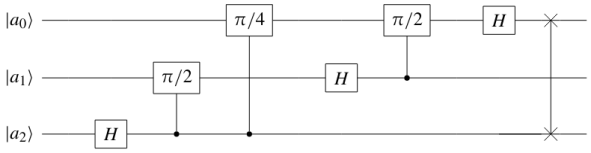

Figure 3: Circuit of the QFT for 3 qubits [1]

IV. QFT - CODE

Now after discussing the theory behind DFT and how to get to the QFT circuit via a decomposition of the DFT matrix, we can start implementing our own QFT algorithm. My code consists of a QFT algorithm written from scratch and only using single qubit gates, the DFT matrix and a short script to test both algorithms and compare to qiskit's implementation of the QFT.

The DFT code is quite short, because it only consists of a loop creating the matrix in 2. QFT code however is a bit bigger and I would like to take some time to go through my thought process. I started to program top to bottom, if you can say it like this. I mean, I immediately started with the QFT circuit implementation and as soon, as I realized, the function becomes to complex or can be divided into smaller steps, I created functions for these problems. For example looking at the QFT function it only consists of the loops for the hadamard gate for each qubit and the multiple rotation gates, that come after each hadamard. How the gates are created is the task of another function. To make the kronecker products easier in the loops, my idea was to store gate operations as strings and then pass them to a function kronString() that turns a string into a kronecker product and returns the gate.

The Hadamard and the Cphase functions create the gates that appear in the QFT circuit. In the end, there are the SWAP, to reverse the qubit order. I spent quite a lot of time on them, because I didn't understand how to build n-qubit SWAP gates from scratch. This could be solved, by the decomposition of a single SWAP gate into three CX gates, for example $\text{SWAP}_{2,4} = \text{CX}_{4,2}\ \text{CX}_{2,4}\ \text{CX}_{4,2}$. So this is why I have added the functions reverseOrder(), swap() and CNOT(). To test these functions, especially the reversOrder() function, I have added functions to transform states from basis state representation to binary representation, because in the binary representation you can easily check the SWAP operations by eye.

Interesting thought about SWAP gates is, that there implementation should be closely connected to qubit chip architecture, because there are physical limitations in swapping qubit states. It is better to swap adjacent qubits then two that lie on completely opposite sides of the chip.

A. QFT - Results

In the TestingQFT.ipynb the QFT got tested. The idea was to create a list with some qubit numbers and for each qubit number create some states, calculate the DFT of the state and compare it to the results from the DFT matrix and the QFT from qiskit. In fig.4 there is a histogram of how well my algorithm performed for some qubit numbers. In an earlier

stage, the QFT only worked for 2 qubits and sometimes for three qubits. This is why I originally added this plot. Now that it works perfectly, the plot does not really make sense to show, I would like to keep it however, just because I already had it.

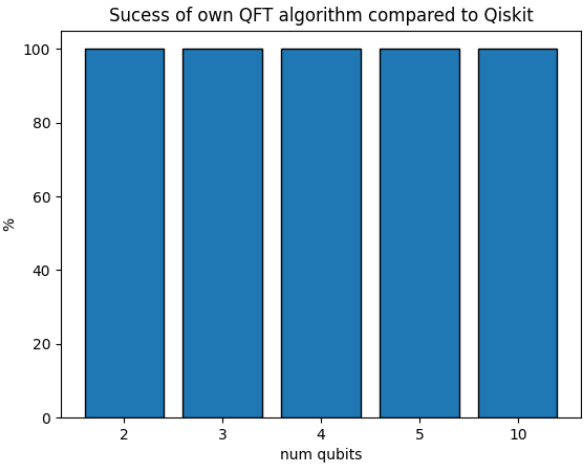

Figure 4: Performance of own QFT algorithm compared to qiskit's build-in QFT function

V. Phase Estimation

Let's have a look at a more complex algorithm, that utilises the QFT. The phase estimation algorithm is desgigned to calculate the eigenvalues of a unitary matrix. The circuit itself is divided into two parts.

1) 1. Apply controlled gates with powers of the unitary matrix to some qubits in a way, that creates the same result as applying the QFT to them.
2) 2. Apply the inverse QFT to retrieve the phase.

The most important prerequisite for this to work, is that unitary matrices have eigenvalues with absolute value of 1. This can be seen easily by using the unitarity relation and eigenvalue equation. Take a unitary operator U and an eigenstate $|x\rangle$ of that operator. Then we have

$$U|x\rangle = \lambda|x\rangle$$

Multiplying the equation with the dual space representation of itself we get

$$1 = \langle xU^\dagger|Ux\rangle = |\lambda^2|$$

Because λ has absolute value of 1 we can express it through one parameter ϕ in the form of

$$\lambda = \exp\{2\pi i\phi\}$$

The accuracy of the phase estimation is limited to the number of fractional bits (that's why it is called phase **estimation**). This becomes clearer when we try to derive the circuit.

A. Example of Phase Estimation with one qubit

Let's suppose we start with a unitary operator U and one of it's eigenvectors $|u\rangle$ and eigenvalue $\exp\{2\pi i0.\phi_0\}$. We divide our complete quantum register into a number of t computational bits, on which we perform the estimation and n bits, in which we store the eigenstate of U. Additionally assume that U can be written as

$$U = \begin{pmatrix} 1 & 0 \\ 0 & \exp\{2\pi i0.\phi_1\} \end{pmatrix}$$

We follow the derivation in [1] and look at first at the circuit in fig. 5. After the first Hadamard gate the state $|\psi_1\rangle$ representing the full quantum register reads

$$|\psi_1\rangle = |+\rangle \otimes |\psi\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle \otimes |\psi\rangle + |1\rangle \otimes |\psi\rangle\right)$$

putting the first qubit into superposition. After the controlled-U gate the total state becomes

$$\begin{aligned} |\psi_2\rangle &= \frac{1}{\sqrt{2}}\left(|0\rangle \otimes |\psi\rangle + |1\rangle \otimes U|\psi\rangle\right) \\ &= \frac{1}{\sqrt{2}}\left(|0\rangle \otimes |\psi\rangle + |1\rangle \otimes \exp\{2\pi i0.\phi_0\}|\psi\rangle\right) \\ &= \frac{1}{\sqrt{2}}\left(|0\rangle + \exp\{2\pi i0.\phi_0\}|1\rangle\right) \otimes |\psi\rangle \end{aligned}$$

Here comes the importance into play, that the lower register is set to an eigenstate of U. This means, that the lower register doesn't change and we can then kick the phase to the uper register, by factoring out the lower state. Right now, if we were to measure our computational qubit, it is either in state $|0\rangle$ or $|1\rangle$ with equal propability, independent of ϕ_0. This is because, both amplitudes of the basis states have absolute value 1, making them equally propable, no matter the value of ϕ_0. Therefore we apply the Hadamard gate again, to kind of reverse the superposition we created before. The final state of the computational register then reads

$$|\psi_3\rangle = \frac{1}{2}\left(1 + \exp\{2\pi i0.\phi_0\}\right)|0\rangle + \frac{1}{2}\left(1 - \exp\{2\pi i0.\phi_0\}\right)|1\rangle$$

A short calculation reveals, that for $\phi_0 = 0$ the state collapses to $|0\rangle$ and for $\phi_0 = 1$ to $|1\rangle$. Note that for one qubit only the QFT and its inverse both come down to just the Hadamard gate!

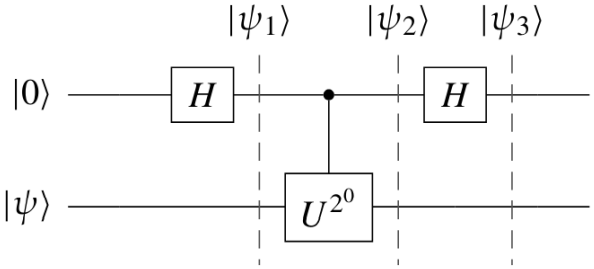

Figure 5: Building block of the phase estimation circuit. $|0\rangle$ is the operational qubit and $|\psi\rangle$ are the ones keeping the eigenstate of U [1]

B. Phase Estimation with two qubits

On our path to find a general phase estimation algorithm with arbitrary accuracy, we increase the accuracy now to two qubits, meaning we can calculate phases that are either 0, 0.25, 0.5 or 0.75. We follow the schematic in fig. 6 and start evaluating the state of our total register after the first controlled U gate.

$$|\psi_1\rangle = \frac{1}{\sqrt{2^2}}\left(|0\rangle + \exp\{2\pi i 0.\phi_0\phi_1\}\right)|1\rangle \otimes \left(|0\rangle + |1\rangle\right) \otimes |\psi\rangle$$

Before we apply the second controlled U gate, let's examine first, what exponentiating it, does to our phase. Squaring means we apply U two times, so we get a rotation with double the angle

$$U^2|\psi\rangle = \exp\{2\pi i 2\phi)\}|\psi\rangle$$

But what does really happen to the phase? A closer look at the fractional representation gives us a better insight

$$\begin{aligned} 2\phi &= 2(0.\phi_0\phi_1) \\ &= 2\left(\phi_0 2^{-1} + \phi_1 2^{-2}\right) \\ &= \phi_0 + \phi_1 2^{-1} = \phi_0.\phi_1 \end{aligned}$$

The decimal point got shifted one to the right, which has some drastic consequences, when looking at the amplitudes of the states.

$$\begin{aligned} \exp\{2\pi i 2\phi\} &= \exp\{2\pi\phi_0.\phi_1\} \\ &= \exp\{2\pi i(\phi_0 + 0.\phi_1\} \\ &= \exp\{2\pi i\phi_0\}\exp\{2\pi i(0.\phi_1)\} \end{aligned}$$

ϕ_0 can only be 0 or 1, which means, that the first exponential is always 1. So the binary fraction reduces to only that position which is equal to the exonentiation of U. In general

$$\exp\big\{2\pi i(2^k\phi)\big\} = \exp\{2\pi i 0.\phi_k\}$$

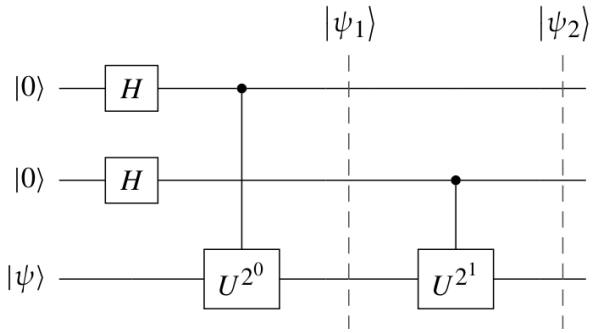

Figure 6: Phase estimation building block with two qubits. Notice the exponentiation of the controlled U gate [1]

After applying U^2 our state is therefore

$$|\psi_2\rangle = \frac{1}{\sqrt{2^2}}\left(|0\rangle + \exp\{2\pi i 0.\phi_0\phi_1\}\right)|1\rangle \tag{5}$$

$$\otimes \left(|0\rangle + \exp\{2\pi i 0.\phi_1\}|1\rangle\right) \otimes |\psi\rangle \tag{6}$$

Comparing this to 4, we notice, that applying the exponentiated U gates to our computational register is the same as applying the QFT. The difference is, that our fractional representation, now has some physical meaning, because it contains information about the phase of the eigenvalue belonging to $|\psi\rangle$. This becomes clear, when applying the inverse QFT circuit to retrieve the binary phase representation

$$\mathrm{DFT}_{0,1}|\psi_2\rangle = |\phi_1\rangle \otimes |\phi_0\rangle \otimes |\psi\rangle \ .$$

C. General Phase Estimation Algorithm

From this we can deduce a general algorithm with a t computational qubits. From 6 we see, that the 0th power of 2 should be applied to the last qubit in the computational register, to match the QFT definition. The state of the t register before the inverse QFT is then

$$\begin{aligned} &\frac{1}{2^{t/2}}\left(|0\rangle + \exp\big\{i\pi 2^{t-1}\phi\big\}|1\rangle\right) \\ &\otimes \frac{1}{2^{t/2}}\left(|0\rangle + \exp\big\{i\pi 2^{t-2}\phi\big\}|1\rangle\right) \\ &\vdots \\ &\otimes \frac{1}{2^{t/2}}\left(|0\rangle + \exp\big\{i\pi 2^{t-0}\phi\big\}|1\rangle\right) \end{aligned}$$

with

$$\phi = 0.\phi_{t-1}\phi_{t-2}...\phi_0$$

It is important to keep the ordering of the qubits in mind. Now we reversed the order from the two qubit phase estimation 6, because our state before the inverse QFT is

$$\frac{1}{2^{t/2}}\left(|0\rangle + \exp\{2\pi i 0.\phi_{t-1}\}|1\rangle\right) \tag{7}$$

$$\otimes \frac{1}{2^{t/2}}\left(|0\rangle + \exp\{2\pi i 0.\phi_{t-1}\phi_{t-2}\}|1\rangle\right) \tag{8}$$

$$\vdots \tag{9}$$

$$\otimes \frac{1}{2^{t/2}}\left(|0\rangle + \exp\{2\pi i 0.\phi_{t-1}...\phi_0\}|1\rangle\right) \tag{10}$$

$$\tag{11}$$

I kind of didn't want to reverse the order, because it can get you confused quite fast, but the final circuit schematic is used for that reversed ordering, which is why I have to write it like this. The full schematic of the phase estimation algorithm with an accuracy of t qubits is shown in fig. 7

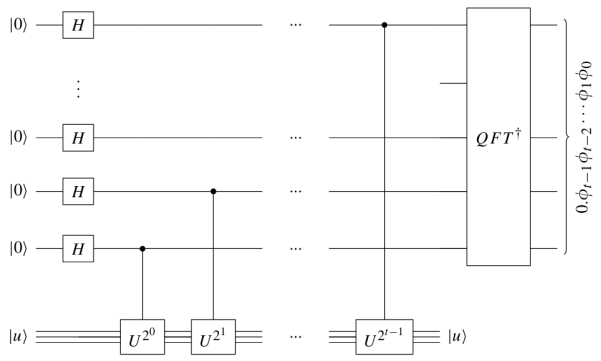

Figure 7: Full phase estimation algorithm. The inverse QFT turns the phase of the eigenvalue into approximated binary representation in the t register [1].

D. Phase Estimation - Implementation

The implementation of the phase estimation algorithm took me some while and I didn't really manage to get any good results with it. The general procedure for implementing the functions however was again top to bottom.

- A main function as a box for the algorithm and its subroutines
- A function that applies all the Hadamard gates to the upper registerr
- A function that applies the controlled U gates
- A function for the inverse QFT[2]
- and finally a function that transfers the binary encoded phase to the estimation

Additionally I added/copied from the QFT some helper functions to convert states to binary representation or calculate kronecker products from strings. For a more detailed description of the code and thoughts behind subroutines like loops I refer to the actual code on github[3]

E. Phase Estimation - Results

The phase estimation algorithm works properly for eigenvalues, that can be represented as binary fractions. It can for example calculate an eigenvalue of 0.0625. For this to work however, t has to be 4 at least, as expected. With less qubits, it is impossible to predict. But what happens, if the eigenvalue is something else. [2] describes in chapter 5.2.1 that the difference between the phase estimation prediction can be and the actual eigenvalue can be corrected and derives a formula for the size of the upper register dependent on the dimension of U and to successfully obtain the phase to n bits

$$t = n + \left\lceil \log 2 + \frac{1}{2e} \right\rceil$$

[2]Here I used the inverse QFT Matrix and not the algorithm, to save some computation time.

[3]This time I invested some time in commenting and cleaning up the code where e describes the fail rate. I tried to follow the mathematics and find a similar dependency, however something must be wrong with my phase estimation algorithm. Setting an arbitrary error margin of 5%, an increasing upper register, didn't give a higher propability of finding a phase close to the real one. The propability for a good guess seems to be always around 10% in this case.

Another problem that [2] adresses is the state preparation of the eigenstates of U. This opens up a completely new topic with many things to explore and would be a great topic for another project.

VI. Conclusion

To conclude we have shown in this report the implementation of the QFT algorithm from scratch. Only numpy and single qubit gates and operations, that could theoretically run on a quantum computer, have been used. The QFT algorithm works as expected. After that, the phase estimation algorithm has been implemented from scratch as well. The algorithm manages to estimate phasese representable as binary fractions perfectly. For random eigenvalues, the algorithm has many problems and the average success rate seems to be independent of t register size and, for an arbitrary error margin of 5%, equal to 10%. The accuracy and this behaviour needs more investigating and [2] offers some information to dive deeper into the topic.

Appendix

Acknowledgment

I would like to thank the lecturers Morten Hjorth-Jensen, Ruben Guevara and Keran Chen for this excellent course. I have learned a lot and the self study part was especially good combinable with my Erasmus time schedule.

References

[1] R. Hundt, *Quantum Computing for Programmers*. Cambridge University Press, 2022.

[2] C. Nielsen, *Quantum Computation and Quantum Information*. Cambridge University Press, 2010.