AC20 Report:
Quantum computation and its capacity to create a real random number

Bozane Marius & Trombini Quentin

April 2021

Contents

Part I

Quantum computation and randomness

1 Introduction to Random Number Generation

In most programs nowadays, random numbers are wildly used (from blockchain encryption to video games). Currently, the most popular way to get one is to use a pseudo-random number generator[7]. This method uses a chaos algorithm like Xor-shift[8] that takes a "seed" as input (like the computer clock in milliseconds). This means that even with 2 close seeds, you will have completely different results. The issue here is that the system is fully determined by its initial value, so this is not truly random, and you may exploit it. For example, you may trick a video game that uses this by freezing the computer clock from the game's point of view.

If you want to avoid these exploits (when you work in cryptography, for example), you can use a Hardware Random Generator. This method uses physical events (that can be related to quantum mechanics) to generate a random number. For instance, a nuclear radiation source with a Geiger counter, or thermal noise. This is way harder to exploit from an attacker perspective as you cannot manipulate the system. However, you can still try to predict it, because it may have some bias: you can have the 1s or 0s predominating, thus your RNG is not stable anymore. For example, with the Geiger counter, although you have equal probability at first, you will end up having more 0s and less 1s, because of the decreasing source of radioactivity.

Random number generation is used in cryptography, video games, computer simulation, or gambling. For computer simulation or gambling, it is important that your system has as little bias as possible, so pseudo-random generator is great, but for cryptography, you need to be sure that your seed cannot be precisely known or set, and it is less important to avoid bias. In these cases, a Hardware generator can be mandatory.

That's why we looked at quantum computers, with their q-bits and superposed states, for our AC20. Our goal this semester was to try to generate true random numbers using a quantum computer, and then determine whether this process works as intended .

2 Hadamard gate and superposition state

In a normal computer, everything is 0's and 1's. Those are represented by a difference in voltage within the wires. However, in a quantum computer there are no bits, only q-bits (quantum bits). A q-bit is a normalized (for the hermitian inner product) vector on $\mathbb{C}^2$ in the orthonormal basis $(|0\rangle\,;|1\rangle)$, thus a q-bit is a vector equal to $a\,|0\rangle + b\,|1\rangle$ with $|a|^2+|b|^2 = 1$.

The hermitian inner product is an inner product in a complex vector space where $\overline{z}$ is the complex conjugate of z. Let u and v be two complex vectors over $\mathbb{C}^n$, the hermitian inner product of u and v is: $\langle u, v\rangle = \sum_{i=1}^{n} \overline{u_1} v_1$. The hermitian inner product must satisfy the following properties:

Let (u, v, w) be 3 vectors of $\mathbb{C}^n$ and α a number of $\mathbb{C}$,

- $\langle u + v, w\rangle = \langle u, w\rangle + \langle v, w\rangle$
- $\langle u, \alpha v\rangle = \alpha\langle u, v\rangle$
- $\langle \alpha u, v\rangle = \overline{\alpha}\langle u, v\rangle$
- $\langle u, v\rangle = \overline{\langle u, v\rangle}$
- $\langle u, u\rangle \geq 0$
- $\langle u, u\rangle = 0 \iff u = \vec{0}$

We will now introduce the KET notation:

$$|u\rangle = \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{pmatrix} \tag{1}$$

And its conjugated transpose, the BRA notation:

$$\langle u| = \begin{pmatrix} \overline{u_1} & \overline{u_2} & \dots & \overline{u_n} \end{pmatrix} \tag{2}$$

$$\langle u, v\rangle = \langle u|v\rangle = \sum_{i=1}^{n} \overline{u_1} v_1 \tag{3}$$

At this point, a quantum computer is just a more complex computer, but the whole point of such computing resides in the superposition state. In both quantum and classical computing, logical gates are used to change the value of a bit/q-bit, but quantum computing has some exclusive gates like the Hadamard's gate: when a q-bit goes through this gate it will be in both state 0 and 1. It means that we will have either a 50% chance of measuring a 0 and a 50% chance to get a 1.

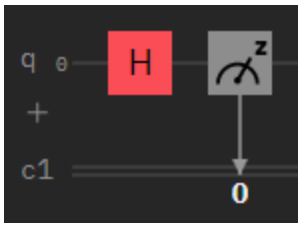
Now let's dive into the physical part to know how the superposition state is working. There are two different types of quantum computer mostly in use: the first one will measure the spin of an electron and the Hadamard gate is an electron beam passing through a magnetic field. After the gate, the spin is deflected towards the north or the south with a probability of $\frac{\sqrt{2}}{2}$.

- A $|0\rangle$ q-bit will become a $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$.
- A $|1\rangle$ q-bit will become a $\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$.

The other type of computer is the one used by IBM. Using the cloud, they are providing free use of their quantum computer using the cloud, but we will come back to this point later. In this type of computer we cool down a metal piece (as low as 40 or even 0,015 Kelvin for IBMQ computers), although there is no tension generator, a small electric flux will travel through the metal piece. Micro-waving this piece will increase its energy level and when the energy reaches the first excitement level the q-bit attains the $|1\rangle$ state. For the superposition level, they just send less micro-waves.

The issue with this quantum process is the sensitivity, anything will create noise and change the result, so you need to isolate your system. they are two solutions: create a vacuum chamber or lower the temperature.

Another important gate is the measurement gate, which will get the q-bits out of the superposition state and return either 0 or 1. In the example below you can see a simple circuit with a q-bit initialized to 0, one Hadamard's gate and one measure gate:



In this circuit the q-bits will pass through a Hadamard gate and go towards a superposition state, then the measurement gate will end the superposition state and return the measure.

3 Cnot gate and entanglement

Hadamard's gate is the most important quantum gate, but there is another very important gate: the Pauli-Xgate or CNot gate. This gate will be used on two different q-bits and will flip the second one (the target q-bit) if and only if the first one (the control q-bit) is in state $|1\rangle$. Mathematically, it can be represented by a 4X4 permutation matrix:

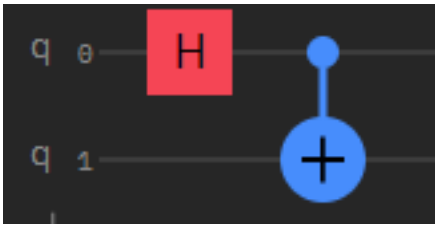
$$CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{4}$$

Let's consider the state of two q-bits q_1 and q_2: q. Then, if we apply the CNot gate:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} a \\ b \\ d \\ c \end{pmatrix} \tag{5}$$

As we can see, it will only flip in cases $|10\rangle$ and $|11\rangle$, in which the control q-bit is equal to 1.

Now if we apply a CNot gate after a Hadamard's gate using the q-bit in superposition as a control q-bit, like in the figure below:



After the Hadamard's gate we will get the following state:

$$|\psi_1\rangle = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ \frac{1}{\sqrt{2}} \\ 0 \end{pmatrix} \tag{6}$$

Then after the application of the CNot gate:

$$|\psi_2\rangle = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ \frac{1}{\sqrt{2}} \\ 0 \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ 0 \\ \frac{1}{\sqrt{2}} \end{pmatrix} = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) \tag{7}$$

After the passage in a CNot gate we say that the two q-bits are entangled and this is an entanglement state.

4 EPR and GHZ

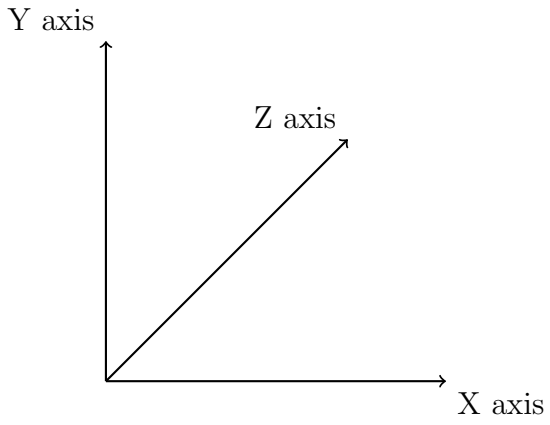
The state used above for the example is called the Bell state or the EPR (Einstein Podolsky Rosen) pairs due to the paradox of the same name about the entanglement state. When one measures two q-bits entangled, the first q-bit will have a probability $\frac{1}{2}$ of either being a 1 or a 0. But the second one will be the same as the first.

It's a problem because if the two q-bits are on the same state either their measurement states are pre-defined or when the first one is measured it send the result to the other one. In case one, the problem is about the superposition state:

How can a q-bit in superposition state have a pre-defined measurement value?

Case two would mean that there is an information travel between the two q-bits faster than light, which is not possible.

John Stewart Bell is an Irish mathematician and physicist who has proven that the global variable theory is false by creating the Bell inequalities.



Assuming we have three directions: X, Y and Z (with 45 degrees between X and Z), as a q-bit is a normalized vector we can measure whether the q-bit is positive in any direction. We can have 8 different possible measures:(+X, +Y, +Z), (+X, +Y, -Z), (+X, -Y, +Z), (+X, -Y, -Z), (-X, +Y, +Z), (-X, +Y, -Z), (-X, -Y, +Z), (-X, -Y, -Z). The inequalities state that if we had a local variable:

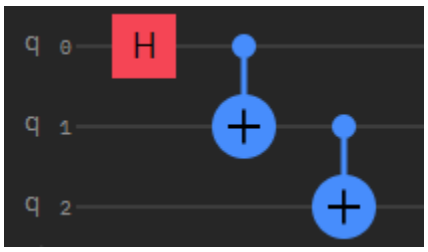
$$P(+X,+Y) \leq P(+Y,+Z) + P(+Z,+X) \tag{8}$$

Using the maths of quantum mechanics, the value of $P(+Y,+Z)$ is equal to $sin^2(\frac{45}{2})$ as the angle between Y and Z is 45 degrees.

$$P(+X,+Y) = sin^2\left(\frac{90}{2}\right) = 0.5, P(+Y,+Z) = P(+Z,+X) = sin^2\left(\frac{45}{2}\right) = 0.23 \tag{9}$$

In quantum mechanics, the Bell inequalities are violated, so there is no hidden variable.

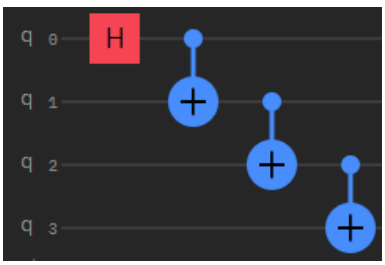
Back to our quantum circuit, there are some derivates of the EPR pairs called the GHZ state. As you can see below, it is a Bell state applied on three q-bits:

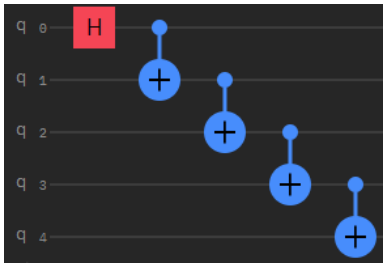


This state is following the same rules as the EPR one:

$$|q_1\rangle \otimes |q_2\rangle \otimes |q_3\rangle = \begin{pmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \end{pmatrix} = a\,|000\rangle + b\,|001\rangle + c\,|010\rangle + d\,|011\rangle + e\,|100\rangle + f\,|101\rangle + g\,|110\rangle + h\,|111\rangle \tag{10}$$

In our situation it will give: $\frac{1}{\sqrt{2}}(|000\rangle + |111\rangle)$ This state got its name from the first person who studied it: Daniel Greenberger, Michael Horne and Anton Zeillinger. We have already seen the derivate for 3 q-bits, but is there a derivate for each number of q-bits? Yes but they don't have a proper name they are called GHZ4, GHZ5, etc. and they follow the same rules as the EPR and GHZ state:





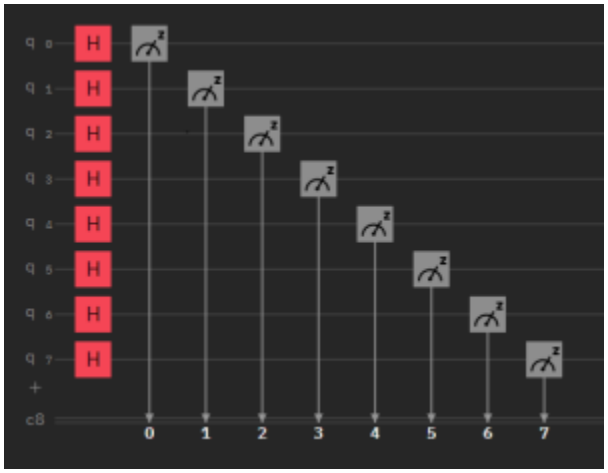
They follow the same computation rules for n q-bits:

$$\frac{1}{\sqrt{2}}(|0\rangle^{\otimes N} + |1\rangle^{\otimes N}) \tag{11}$$

5 First tests and Chi-squared test

5.1 The test with IBM-Quantum Experience

At this point it seems pretty simple to create real randomness, we just have to put a q-bit on a superposition state using a Hadamard gate and measure it. First we used the IBM-quantum experience[2] which grants us free access to little quantum computers (between 1 and 15 q-bits) to create a simple circuit with eight Hadamard's gates and eight measurement gates as you can see below:



To use a quantum computer you can use a composer with a graphical interface, or you can code it using Python. We have chosen the second option, using the Quiskit documentation[5]. We have created a new Jupyter page, and we learned how to create and run a quantum code.

```
%matplotlib inline
# Importing standard Qiskit libraries
from qiskit import QuantumCircuit, execute, IBMQ, ClassicalRegister,
    QuantumCircuit
from qiskit.compiler import transpile, assemble
from qiskit.tools.jupyter import *
from qiskit.visualization import *
from qiskit.providers.ibmq import least_busy
from ibm_quantum_widgets import *
from numpy import pi

# Loading your IBM Q account
provider = IBMQ.save_account('my_token',overwrite=True)
provider = IBMQ.get_provider(hub='ibm-q')
backend = least_busy(provider.backends(filters=lambda x:
    x.configuration().n_qubits >= 8 and not x.configuration().simulator and
    x.status().operational==True))
print("least busy backend: ", backend)

qc=QuantumCircuit(8,8)
```

```
qc.h(0)
qc.h(1)
qc.h(2)
qc.h(3)
qc.h(4)
qc.h(5)
qc.h(6)
qc.h(7)
qc.barrier(range(8))
qc.measure(range(8),range(8))

job = execute(qc, backend, shots=8196)
result = job.result()
counts=result.get_counts(qc)
print(counts)
```

The first tricky part is the connection to a quantum computer: after connecting to provider using your token, you need to get all the usable providers including the simulator, and only then can you connect to a backend. Our code was connecting to the least busy backend, thus we didn't have to wait too long between each test.

Then is the circuit part: we created a circuit with eight q-bits and eight bits, we applied a Hadamard gate to all the q-bits using the qc.h() function, and then we created a measurement of all the system. We chose to launch the program 8196 times, so we get more data and be more precise.

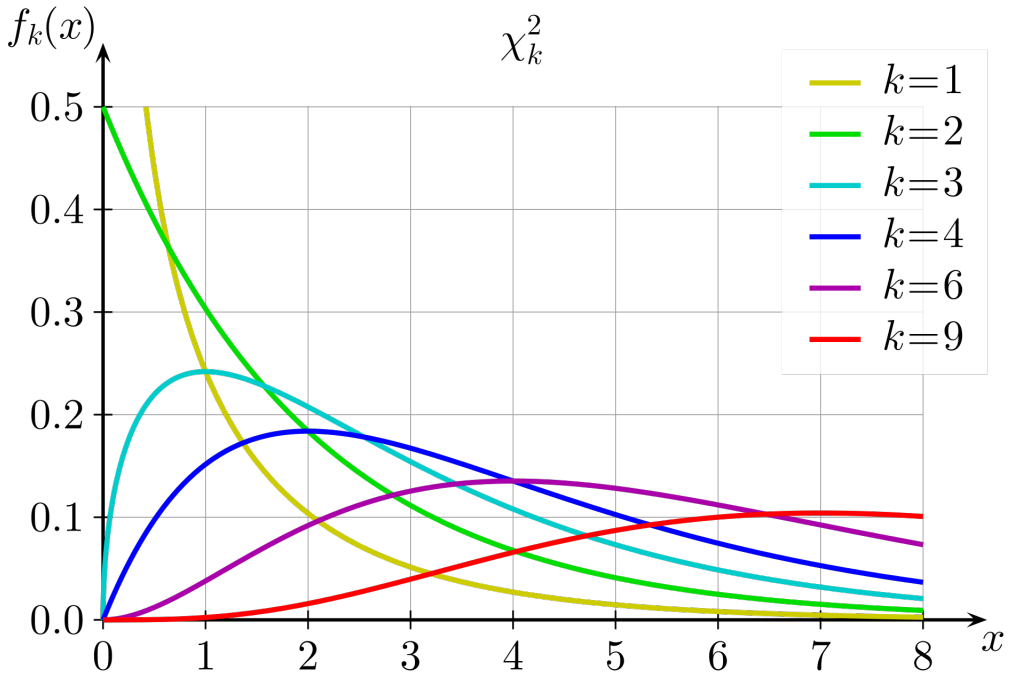
5.2 Chi-squared explanation

The χ^2 test is a statistical test that allows us to evaluate how likely our results would be, assuming our null hypothesis is true. It uses the Chi-square probability density function[4][6]:

$$f_\chi(x;k) = \frac{1}{2^{\frac{k}{2}}\Gamma(\frac{k}{2})} x^{\frac{k}{2}-1} e^{-\frac{x}{2}} \tag{12}$$

Where k is our degree of freedom, and $\Gamma(\frac{k}{2})$ the Gamma function:

$$\Gamma(z) = \int_0^{+\infty} t^{z-1} e^{-t} dt \tag{13}$$



It is important to note that

$$\forall k \in \mathbb{N}^{+*}, \quad \int_0^{+\infty} f_\chi(x;k)dx = 1 \tag{14}$$

In our χ^2 test (a Pearson's test), we do n measures on k q-bits. Let H_0 be: "the probability that there is i q-bits with a state of 1 is equal to p_i". $\forall i \in [\![0,k]\!]$, let x_i be the observed number and m_i be the expected number. We then have:

$$\forall i \in [\![0,k]\!], \qquad m_i = n * p_i \tag{15}$$

$$\sum_{i=0}^{k} x_i = n \sum_{i=0}^{k} p_i = \sum_{i=0}^{k} m_i = n \tag{16}$$

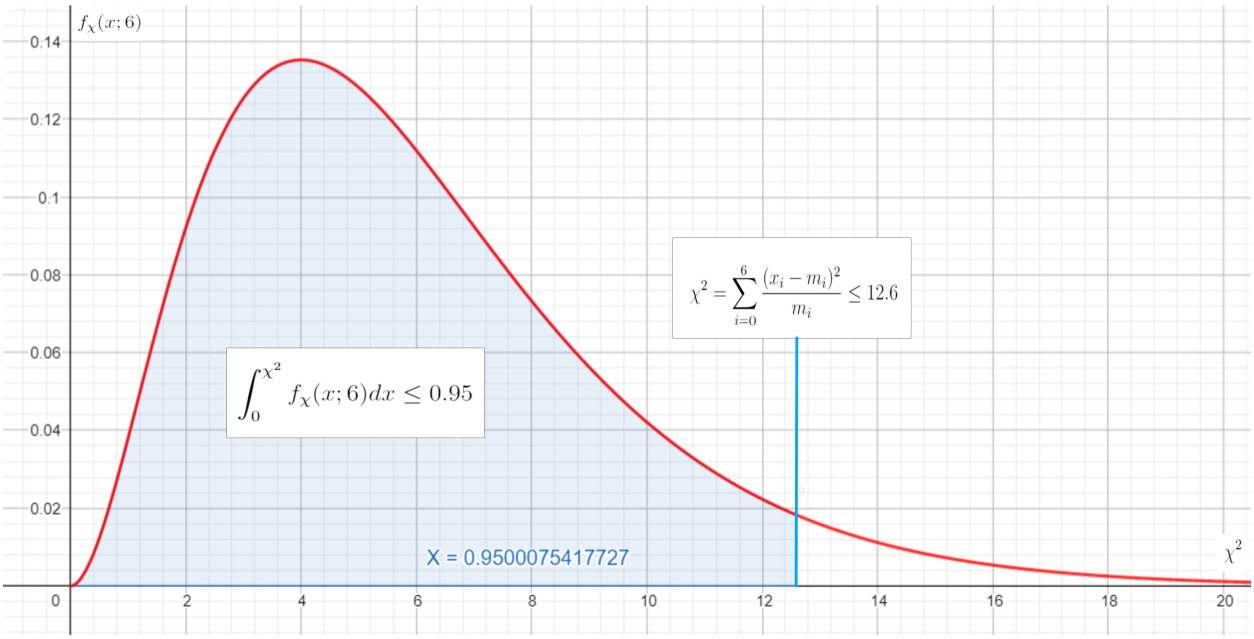
We chose to reject the null hypothesis H_0 if the result is in the 5% most extreme values of the χ^2. Let χ^2 be:

$$\chi^2 = \sum_{i=0}^{k} \frac{(x_i - m_i)^2}{m_i} \tag{17}$$

To accept H_0, we need to have:

$$\int_0^{\chi^2} f_\chi(x; k)dx \leq 0.95 \tag{18}$$

This is easier to understand with this graph, that represents the chi-squared probability density for $k = 6$:



As you can see, for $k = 6$, the null hypothesis is rejected if $\chi^2 > 12.6$

5.3 Analysing our first results

At first, we have done our test with 8 q-bits (so 8 degrees of liberty). Under our null hypothesis, p_i follows a binomial distribution, with:

- $p = 0.5$ (because the q-bit state should be $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$)
- n = 8196
- $p_i = \mathbb{P}(X = i) = \binom{n}{i} * p^i * (1-p)^{n-i}$

Here are our first results:

i	m_i	x_i	$\frac{(x_i - m_i)^2}{m_i}$
0	32	43	3.78
1	256	387	67.03
2	896	1169	83.18
3	1792	2037	33.50
4	2240	2182	1.50
5	1792	1549	32.96
6	896	642	72.00
7	256	175	25.63
8	32	8	18.00
χ^2			337.58

i	m_i	x_i	$\frac{(x_i - m_i)^2}{m_i}$
0	32	58	21.125
1	256	388	68.06
2	896	1114	53.04
3	1792	2053	38.01
4	2240	2213	0.33
5	1792	1537	36.29
6	896	639	73.72
7	256	168	30.25
8	32	22	3.125
χ^2			323.95

$$\int_0^{323.94} f_\chi(x; 8)dx = 0.999... > 0.95 \tag{19}$$

With these tests, it is **more than extremely unlikely** that H_0 is true. We now have to find a new H_0 that is more likely to be true.

After counting the average for each q-bit, we found that p should be around 0.535. We repeated the test with this new value:

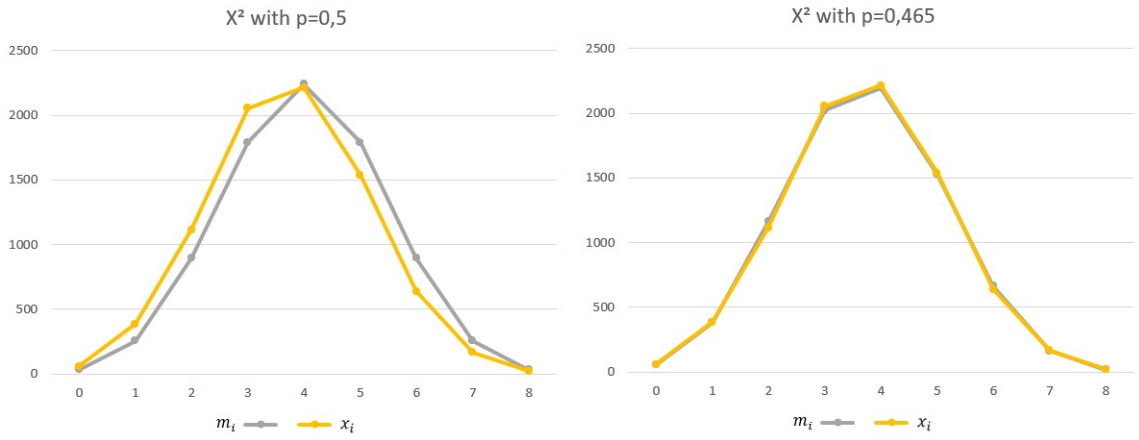
i	m_i	x_i	$\frac{(x_i - m_i)^2}{m_i}$
0	32	43	2.61
1	256	387	0.06
2	896	1169	0.03
3	1792	2037	0.12
4	2240	2182	0.09
5	1792	1549	0.31
6	896	642	0.71
7	256	175	0.63
8	32	8	5.48
χ^2			10.04

i	m_i	x_i	$\frac{(x_i - m_i)^2}{m_i}$
0	32	58	0.16
1	256	388	0.08
2	896	1114	2.06
3	1792	2053	0.49
4	2240	2213	0.12
5	1792	1537	0.06
6	896	639	0.92
7	256	168	0.06
8	32	22	0.94
χ^2			4.9

$$\int_0^{10.04} f_\chi(x;8)dx = 0.74 < 0.95 \tag{20}$$

$$\int_0^{4.9} f_\chi(x;8)dx = 0.23 < 0.95 \tag{21}$$

With these results, we can accept H_0.



To confirm our results, we re-launched five tests, and for each one, we can accept H_0 with $p = 0.465$ but the null hypothesis with $p = 0.5$ is always rejected. Generating a random number with an equal probability was not as simple as we think using quantum computing. To be sure that our code was not the issue, we launched it on a simulator provided by IBMQ. The results on the simulator were almost perfect: it passed the χ^2 test with $p = 0.5$ every time. Thus, we can say the issue came from the quantum computer.

5.4 Tests on $\sqrt{X}$ and $\sqrt{X}^\top$ gates

The Hadamard gate is not the only gate that should allow us to put the q-bit in an equal superposition state, and then get a random draw. We have done that on 7 q-bits, with two gates: $\sqrt{X}$ and $\sqrt{X}^\top$. We should get this state:

$$|q_0\rangle \otimes |q_1\rangle \otimes |q_2\rangle \otimes |q_3\rangle \otimes |q_4\rangle \otimes |q_5\rangle \otimes |q_6\rangle = \frac{1}{\sqrt{128}} \sum_{i=0}^{127} |i\rangle \tag{22}$$

Because the χ^2 tests have always rejected the null hypothesis of having a probability of measuring 50% of 1's, we now try to approximate the real probability of measuring a 1. To do that, we use a confidence interval computed at the 95% confidence level. With N measures, S q-bits measured with a state of 1, we try to find the true probability p. Each individual q-bit should be in a $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$, so p should be around 0.5. We can say that in 95% of the tests:

$$\frac{S}{N} - 1.96\sqrt{\frac{(S/N)(1-(S/N))}{N}} \quad < \quad p \quad < \quad \frac{S}{N} + 1.96\sqrt{\frac{(S/N)(1-(S/N))}{N}} \tag{23}$$

Here are our results for $\sqrt{X}$:

$$N = 8192$$

q_i	S	S/N	$1.96\sqrt{\frac{(S/N)(1-(S/N))}{N}}$
0	3803	0.464	0.0108
1	3834	0.468	0.0108
2	4066	0.496	0.0108
3	2857	0.349	0.0103
4	4024	0.491	0.0108
5	3761	0.459	0.0108
6	3801	0.464	0.0108
$0.454 < p < 0.501$			

q_i	S	S/N	$1.96\sqrt{\frac{(S/N)(1-(S/N))}{N}}$
0	3822	0.467	0.0108
1	3823	0.467	0.0108
2	3978	0.486	0.0108
3	3457	0.422	0.0106
4	4009	0.490	0.0108
5	3648	0.445	0.0108
6	3878	0.473	0.0108
$0.435 < p < 0.500$			

As you can see, results are really inconsistent, depending on the q-bit. The q_3 q-bit looks "broken", with its very low probability of returning a 1. Some q-bits seem to achieve the goal of the 50% probability, but it is a minority.

The results for $\sqrt{X}^\top$:

$$N = 8192$$

q_i	S	S/N	$1.96\sqrt{\frac{(S/N)(1-(S/N))}{N}}$
0	3864	0.472	0.0108
1	3907	0.477	0.0108
2	3961	0.484	0.0108
3	3638	0.444	0.0108
4	4010	0.490	0.0108
5	3816	0.466	0.0108
6	3767	0.460	0.0108
$0.433 < p < 0.500$			

q_i	S	S/N	$1.96\sqrt{\frac{(S/N)(1-(S/N))}{N}}$
0	3759	0.459	0.0108
1	3856	0.471	0.0108
2	4053	0.489	0.0108
3	3569	0.440	0.0107
4	4019	0.491	0.0108
5	3848	0.470	0.0108
6	3775	0.461	0.0108
$0.425 < p < 0.506$			

The results here are similar, with some q-bits near 50%, even if there are still noticeable variations depending on the q-bit.

5.5 EPR and GHZ tests

We have recreated the EPR and GHZ states using one Hadamar's gate and some CNOT gates.

i	x_i	p_i	$p_i \setminus \{noise\}$
00	3818	46,61%	50,70%
01	370	4,52%	
10	292	3,56%	
11	3712	45,31%	49,30%
Test 1 noise = 8.08%			

i	x_i	p_i	$p_i \setminus \{noise\}$
00	3668	44,78%	48,71%
01	335	4,09%	
10	326	3,98%	
11	3863	47,16%	51,29%
Test 2 noise = 7.41%			

i	x_i	p_i	$p_i \setminus \{noise\}$
00	3668	44,78%	48,71%
01	335	4,09%	
10	326	3,98%	
11	3863	47,16%	51,29%
Test 3 noise = 8.07%			

i	x_i	p_i	$p_i \setminus \{noise\}$
00	3699	45,15%	49,35%
01	337	4,11%	
10	359	4,38%	
11	3797	46,35%	50,65%
Test 4 noise = 8.50%			

i	x_i	p_i	$p_i \setminus \{noise\}$
00	3692	45,07%	49,06%
01	347	4,24%	
10	320	3,91%	
11	3833	46,79%	50,94%
Test 5 noise = 8.14%			

i	x_i	p_i	$p_i \setminus \{noise\}$
00	3730	45,53%	49,50%
01	352	4,30%	
10	305	3,72%	
11	3805	46,45%	50,50%
Test 6 noise = 8.02%			

As we can see on these tests, there is around 8% of "noise" (we call noise the states that should be impossible, $|01\rangle$ and $|10\rangle$ here). When we look at the probability of $|00\rangle$ and $|11\rangle$ afterward, without taking noise into account, we find results that are more acceptable tan previously, with this time a little predominance of the 1's.

The results with different GHZ states are not as good as EPR :

GHZ :

i	x_i	p_i	$p_i \setminus \{noise\}$
000	455	0,456827309	0,538461538
001	16	0,016064257	
010	16	0,016064257	
011	3	0,003012048	
100	10	0,010040161	
101	42	0,042168675	
110	64	0,064257028	
111	390	0,391566265	0,461538462
Test 1 noise = 15.16%			

i	x_i	p_i	$p_i \setminus \{noise\}$
000	476	0,46484375	0,536036036
001	19	0,018554688	
010	5	0,004882813	
011	23	0,022460938	
100	13	0,012695313	
101	20	0,01953125	
110	56	0,0546875	
111	412	0,40234375	0,463963964
Test 2 noise = 13.28%			

There is more noise, and results are far from the 50

GHZ_5 :

i	x_i	p_i	$p_i \setminus \{noise\}$
0	3762	0,464673913	0,59393748
1	38	0,004693676	
10	7	0,000864625	
11	7	0,000864625	
100	56	0,006916996	
101	12	0,001482213	
110	10	0,001235178	
111	64	0,007905138	
1000	91	0,011240119	
1001	2	0,000247036	
1010	4	0,000494071	
1011	39	0,004817194	
1100	4	0,000494071	
1101	30	0,003705534	
1110	37	0,004570158	
1111	437	0,053977273	
10000	105	0,012969368	
10001	3	0,000370553	
10010	3	0,000370553	
10011	5	0,000617589	
10100	4	0,000494071	
10101	9	0,00111166	
10110	8	0,000988142	
10111	126	0,015563241	
11000	57	0,007040514	
11001	14	0,001729249	
11010	22	0,002717391	
11011	183	0,022603755	
11100	10	0,001235178	
11101	169	0,020874506	
11110	206	0,025444664	
11111	2572	0,053977273	0,40606252
Test 1 noise = 21.76%			

i	x_i	p_i	$p_i \setminus \{noise\}$
0	3810	0,470602767	0,602371542
1	50	0,006175889	
10	12	0,001482213	
11	4	0,000494071	
100	55	0,006793478	
101	19	0,002346838	
110	1	0,000123518	
111	49	0,006052372	
1000	75	0,009263834	
1001	1	0,000123518	
1010	7	0,000864625	
1011	21	0,002593874	
1100	5	0,000617589	
1101	34	0,004199605	
1110	38	0,004693676	
1111	452	0,05583004	
10000	78	0,009634387	
10001	3	0,000370553	
10010	1	0,000123518	
10011	7	0,000864625	
10100	2	0,000247036	
10101	15	0,001852767	
10110	7	0,000864625	
10111	150	0,018527668	
11000	55	0,006793478	
11001	9	0,00111166	
11010	20	0,002470356	
11011	198	0,024456522	
11100	11	0,001358696	
11101	184	0,022727273	
11110	208	0,0256917	
11111	2515	0,05583004	0,397628458
Test 2 noise = 21.88%			

The GHZ_5 is even more broken, with a 60% probability of returning the $|00000\rangle$ state, and 21% of noise.

6 Conclusion

At this point we tried several experiments on several computers, and we can generate random numbers, but they are not equally probable. We then investigated our results to figure out what was the issue and why we couldn't generate random numbers properly. In our results we found something weird in each test, as there was a weird q-bit which generated a significantly lower number than the others, and this q-bit is lowering our statistics.

Later we found that our best result (by best we mean the closest to the equally probable distribution) was in a program we launched just after a computer maintenance. Even then there were at least 6000 programs running before ours, so we think the issue is hardware related due to the noise because the computers are not exactly at 0 Kelvin.

Thanks to Mr.Holweck [1], we found a society named ID'quantique[3], selling little electronic chips for smartphones and computers. These chips are small quantum computers which generate random numbers and send it to the device thereafter. We wanted to try whether this computer had the same bias, but unfortunately the starting price was at 1500$ therefore we couldn't try this by ourselves.

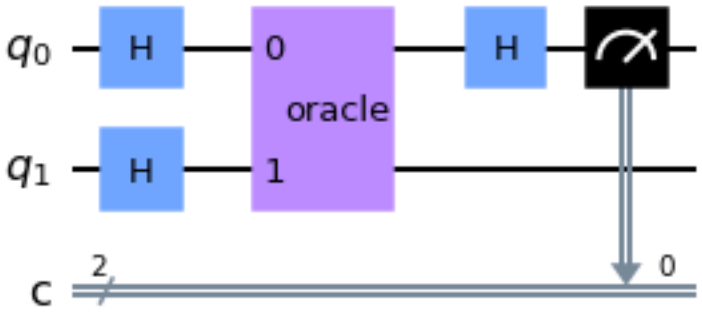
Part II

More about quantum computation

7 Deutsch Josza's algorithm

During our research we have found a lot of things about quantum computing which are not directly related to our subject. This section contains some of this research because we felt it was worthy of interest.

The Deutsch Josza's algorithm is in fact an improvement of the Deutsch algorithm, so first we are going to look at the algorithm created by David Deutsch in 1985. This algorithm checks whether a boolean function $f : \{0, 1\} \to \{0, 1\}$ is constant or balanced. In classical computing you need to call the function twice to check if it's constant or balanced, while in quantum computing we only need one call.



Just above you can see the Deutsch algorithm circuit: it's a two q-bits system which begins in state $|0\rangle |1\rangle$, lets them pass through a Hadamard's gate. An oracle will then push the result through another Hadamard's gate, but only to the first q-bit before the measure. Let's analyse this circuit. After the first Hadamard's gate, we have:

$$|\psi_1\rangle = \frac{1}{2}(|0\rangle + |1\rangle)(|0\rangle - |1\rangle) \tag{24}$$

We then apply the oracle. An oracle is a black box with some mathematical functions inside. In our case, it will transform $|x\rangle |y\rangle$ into $|x\rangle |f(x) \oplus y\rangle$ with $\oplus$ the XOR logical function. So after the oracle we obtain:

$$|\psi_2\rangle = \frac{1}{2}(|0\rangle (|f(0)\rangle \oplus 0) - |f(0)\rangle \oplus 1)) + |1\rangle (|f(1)\rangle \oplus 0) - |f(1)\rangle \oplus 1)) \tag{25}$$

Which led us to:

$$|\psi_2\rangle = \frac{1}{2}(-1^{f(0)} |0\rangle (|0\rangle - |1\rangle) + -1^{f(1)} |1\rangle (|0\rangle - |1\rangle) \tag{26}$$

$$|\psi_2\rangle = \frac{1}{2}(|0\rangle + (-1)^{f(0)\oplus f(1)} |1\rangle)(|0\rangle - |1\rangle) \tag{27}$$

Finally we recognize the state of the second q-bit in a superposition state. We are now only interested in the first q-bit, currently in the following state:

$$|\psi_2\rangle = \frac{1}{\sqrt{2}}(|0\rangle + (-1)^{f(0)\oplus f(1)}\,|1\rangle) \tag{28}$$

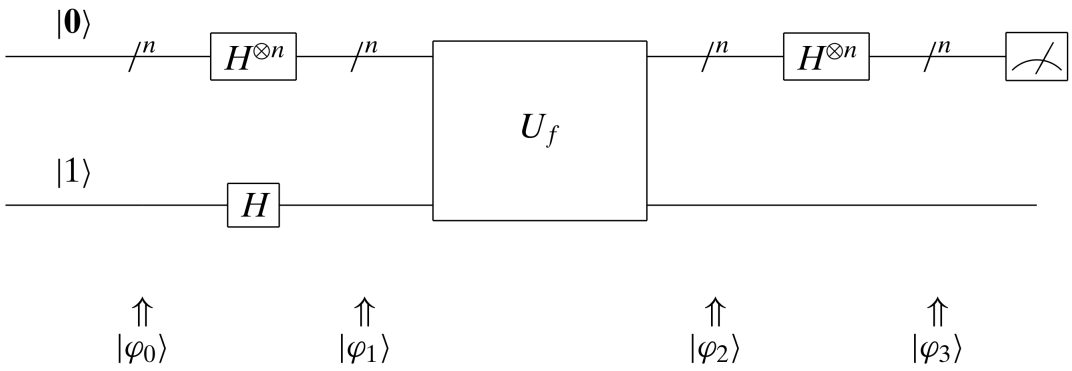
Now if the function is continuous $f(0) = f(1)$ and the XOR operator returns a 0, our q-bit will go into:

$$|\psi_2\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \tag{29}$$

If we apply a Hadamard gate on this we will get $|0\rangle$. In the other hand, when $f(0) \neq f(1)$ we have:

$$|\psi_2\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \tag{30}$$

This will give us $|1\rangle$ after the passage in the Hadamard's Gate. Henceforth, this algorithm will return 0 if the function is constant and 1 otherwise. The problem with this algorithm is its specificity: it only works for $f : \{0, 1\} \to \{0, 1\}$. Josza later worked with Deutsch, and they improved the algorithm to work on $f : \{0, n\} \to \{0, 1\}$. They created this algorithm in 1992. There is only a slight difference between the Deutsch and the Deutsch-Josza algorithm diagrams: instead of having one q-bit with $|0\rangle$ and one with $|1\rangle$ we will get n q-bits with $|0\rangle$ ($|0\rangle^{\otimes n}$) and one with $|1\rangle$:



At the state ψ_1, by applying the tensor product on all the q-bits, we get:

$$|\psi_1\rangle = \frac{1}{\sqrt{2^{n+1}}} \sum_{i=0}^{2n-1} |i\rangle\,(|0\rangle - |1\rangle) \tag{31}$$

The oracle is the same as in the Deutsch algorithm. It will transform $|x\rangle\,|1\rangle$ into $|x\rangle\,|f(x) \oplus y\rangle$. At the point $|\psi_2\rangle$ we get:

$$|\psi_2\rangle = \frac{1}{\sqrt{2^{n+1}}} \sum_{i=0}^{2n-1} |i\rangle\,|f(i) \oplus (|0\rangle - |1\rangle)\rangle = \frac{1}{\sqrt{2^{n+1}}} \sum_{i=0}^{2n-1} |i\rangle\,(|f(i)\rangle - |f(i) \oplus 1\rangle) \tag{32}$$

We know that $f(x)$ is either 0 or 1, hence the inside of the sum of our equation can only be $|i\rangle \times -1$ or $|i\rangle \times 1$ depending on the $f(x)$ result, so we can factor the whole equation by:

$$|\psi_2\rangle = \frac{1}{\sqrt{2^{n+1}}} \sum_{i=0}^{2n-1} (-1)^{f(i)} |i\rangle (|0\rangle - |1\rangle) \tag{33}$$

As in the Deutsch algorithm, the last q-bit is in a superposition state, we will therefore ignore this q-bit for the rest of our algorithm:

$$|\psi_2\rangle = \frac{1}{\sqrt{2^n}} \sum_{i=0}^{2n-1} (-1)^{f(i)} |i\rangle \tag{34}$$

Then the n first q-bits pass through a Hadamard's gate assuming that $ij = i_1 j_1 \oplus i_2 j_2 \oplus \ldots i_{n-1} j_{n-1}$, and we get:

$$|\psi_3\rangle = \frac{1}{\sqrt{2^n}} \sum_{i=0}^{2n-1} (-1)^{f(i)} \frac{1}{\sqrt{2^n}} \sum_{j=0}^{2n-1} (-1)^{ij} |j\rangle \tag{35}$$

We can factor it as:

$$|\psi_3\rangle = \frac{1}{2^n} \sum_{i=0}^{2n-1} \sum_{j=0}^{2n-1} (-1)^{f(i)+ij} |j\rangle \tag{36}$$

Now assuming that $|j\rangle = |0\rangle$, we can transform the previous equation:

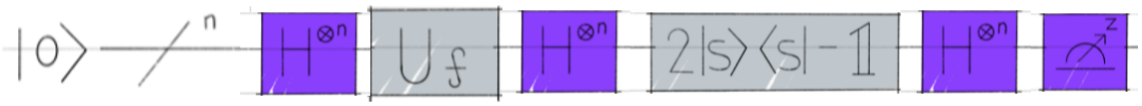
$$|\psi_3\rangle = \frac{1}{2^n} \sum_{i=0}^{2n-1} (-1)^{f(x)} \tag{37}$$

If the function is constant we get $\frac{2^n}{2^n} = 1$. In the other hand, if the function is balanced: $\frac{1}{n^2}((\sum_{i=0}^{2} 1) + (\sum_{i=0}^{2} -1) = 0$. So this algorithm is less complex than a classical algorithm because we only need to call it once, then in classical computation we need between 2 and $\frac{n}{2} + 1$ calls to be sure.

8 Grovers algorithm

Imagine we had a list of N elements, and we want to check the position of a specific element. In a classical algorithm, the optimal choice would be to check each element of the list in turn, and notice when we find the desired one. On average, it will check $\frac{N}{2}$ elements but with a quantum algorithm the desired element can be found in only $\sqrt{N}$ calls, using the Grover algorithm made by Lov Grover in 1996.

The Grover algorithm looks like the following:



This algorithm will start as the Deutsch-Josza with n ($2^n = N$) but without the $|1\rangle$. Then as usual, all the q-bits are going through a Hadamard's gate, so it will give us the following state:

$$|\psi_1\rangle = \frac{1}{\sqrt{2^n}} \sum_{i=0}^{2^n-1} |i\rangle = \frac{1}{\sqrt{2^n}} |\omega\rangle + \frac{1}{\sqrt{2^n}} \sum_{i=0, i\neq\omega}^{2^n-1} |i\rangle \tag{38}$$

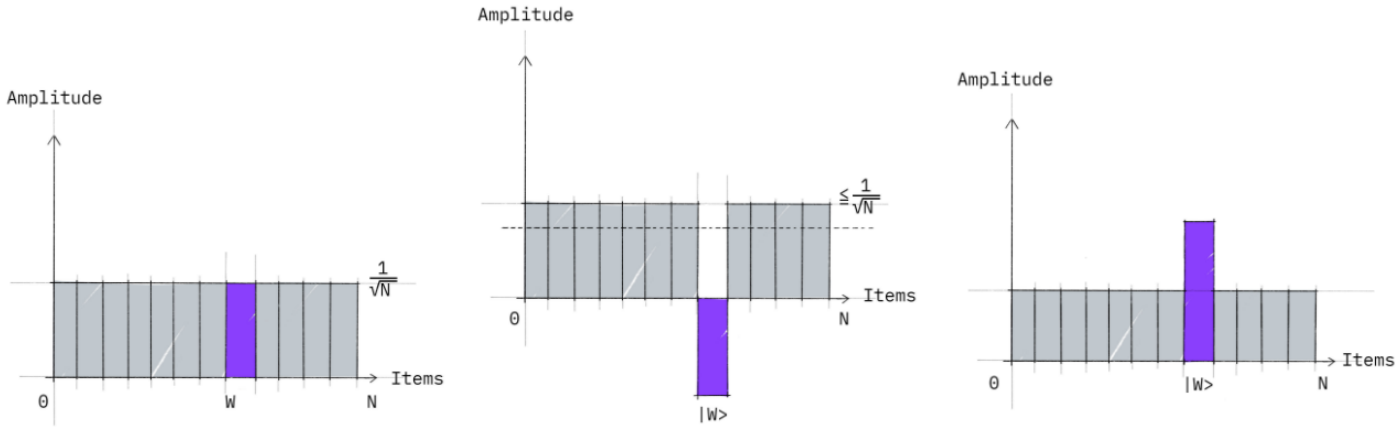
At this point the amplitude of ω and that of the other q-bits are all equal to $\alpha = \frac{1}{2^n}$. Assuming we want to find the element ω, we will use the following function inside an oracle, which is going to associate 1 to $f(\omega)$ and 0 to $f(i)$ for all the other i in the array. It will give us:

$$|\psi_1\rangle = -\frac{1}{\sqrt{2^n}} |\omega\rangle + \frac{1}{\sqrt{2^n}} \sum_{i=0, i\neq\omega}^{2^n-1} |i\rangle \tag{39}$$

After the oracle, we will apply the Grover diffusion operator. It's three logical gates put together; a Hadamard's gate, an oracle equal to $(2\,|0^n\rangle\langle 0^n| - I_n)$ which can be written as: $\begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & -1 & \dots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & -1 \end{pmatrix}$. After this oracle, the third gate is a Hadamard's gate too. This Grover diffusion operator will apply an amplitude inversion by the mean, so the amplitude of ω will increase and the other, decrease:

$$\alpha_i = 2\frac{2^n - 2}{2^{\frac{3n}{2}}} - \frac{1}{2^{\frac{n}{2}}} \quad \text{and} \quad \alpha_\omega = 2\frac{2^n - 2}{2^{\frac{3n}{2}}} + \frac{1}{2^{\frac{n}{2}}} \tag{40}$$

The amplitude of our ω element will increase as such:



And that's all, we are going to repeat the oracle and Grover's diffusion operator approximately $\sqrt{N}$ times, and each time the amplitude of ω will keep increasing. After all these repetitions we can measure the state, and we have a great chance to measure our ω, but to avoid errors this algorithm is generally called 2 times.

We ran this algorithm 8000 times on a computer using IBMQ, Quiskit and the following code:

```
# Importing standard Qiskit libraries
import numpy as np
import matplotlib.pyplot as plt
from qiskit import *
from ibm_quantum_widgets import *
from qiskit.compiler import transpile, assemble
from qiskit.tools.jupyter import *
from qiskit.visualization import *
from qiskit.providers.ibmq import least_busy

# Loading your IBM Q account
IBMQ.save_account('239e6920f40c1c71a95fb58217ecf707322512b8914991c570d52cb07e0d6f6cb01843f9ec2
    overwrite=True)
IBMQ.load_account()
IBMQ.providers()
provider = IBMQ.get_provider(hub='ibm-q', group='open', project='main')
backend = least_busy(provider.backends(filters=lambda x:
    x.configuration().n_qubits >= 2 and not x.configuration().simulator and
    x.status().operational==True))
print("least busy backend: ", backend)

#creating the oracle block
oracle = QuantumCircuit(2,name='oracle')
oracle.cz(0,1)
oracle.to_gate()
```

```
#creating the Grover's diffusion's block
reflection = QuantumCircuit(2,name='reflection')
reflection.h([0,1])
reflection.z([0,1])
reflection.cz(0,1)
reflection.h([0,1])

#initializing our circuit
grover_circ=QuantumCircuit(2,2)

#putting Hadamard's gate, oracle, Grover's diffusion's block and measurement
    altogether in our circuit
grover_circ.h([0,1])
grover_circ.append(oracle,[0,1])
grover_circ.append(reflection,[0,1])
grover_circ.measure([0,1],[0,1])

#launch the program
execute(grover_circ,backend,shots=8000).result().get_counts()
```

It gave us the following results: '00': 51, '01': 292, '10': 191, '11': 7466, with an $\omega = 11$. The algorithm without repetition just returns $\frac{534}{8000} = 0.06675$ false measures. This algorithm is on average pretty accurate. This is a good example on how quantum algorithms can outperform classical ones.

9 Quantum Encryption

Quantum computer are a threat to our encryption, for example the RSA, used mostly by banks, is too long to break for a normal computer, but not for a quantum computer. The Schor's algorithm has a polynomial complexity which can break the RSA. Due to this threat, the encryption must change, and it occurs in two different ways, the post-quantum encryption (trying to create algorithms producing unbreakable encryption code even for quantum computer) and the quantum encryption (using quantum computer).

The second solution led to the BB84 protocol ("created by Bennett and Brassard in 1984"), an encryption way in which, even if someone gets your data, he will never be able to read it. For this protocol, there are three different entities: Alice(A), Bob(B) and Spy(S). Here, A want to send a secret code to B. They need two communication channels: a classical and a quantum one. Both channels don't need to be encrypted or secured.

If A want to send a message to B she will take a string and extract its binary code from it. Then she will randomly choose for each bit if she transforms it into:

- $0 \to |0\rangle, 1 \to |1\rangle$ the classical mode
- $0 \to \frac{|0\rangle+|1\rangle}{\sqrt{2}}, 1 \to \frac{|0\rangle-|1\rangle}{\sqrt{2}}$ the Hadamard mode

A sends all the q-bit to B, he measures the outcome randomly for each q-bit either in classical or Hadamard mode. Then he will tell in which basis he has measured each q-bit. All the q-bits sent and measured in the same basis will become the encryption code. B will then send back to A a small part of the encryption code by the classical channel.

If S has intercepted the quantum transmission, he measures the q-bits and then send different q-bits to B, so due to the superposition state, the q-bits receipted and sent by S are not the same. So when A and B share a little portion of the encryption code, if the two keys are too different they just drop it and repeat the operation until the differences are almost 0. Then A sends the message to B through the classical channel, and he can decrypt it using the encryption code. Even if S intercepts the encoded message by the classical channel. He won't be able to read it as the encryption code is not a mathematical function, but a string generated with an association of random (Hadamard gate) and pseudo-random (Decision on the sending and measuring mode).

10 Appendix

You can find here some additional tests, because it was way too much to be put inside the text:

GHZ_4 :

i	x_i	p_i	$p_i \setminus \{noise\}$
0	3807	0,47023	0,550621927
1	39	0,00482	
10	17	0,00210	
11	19	0,00235	
100	66	0,00815	
101	23	0,00284	
110	12	0,00148	
111	197	0,02433	
1000	188	0,02322	
1001	14	0,00173	
1010	25	0,00309	
1011	220	0,02717	
1100	14	0,00173	
1101	170	0,02100	
1110	178	0,02199	
1111	3107	0,38377	0,449378073
Test 1 noise = 14.60%			

i	x_i	p_i	$p_i \setminus \{noise\}$
0	3811	0,47073	0,550563421
1	26	0,00321	
10	20	0,00247	
11	23	0,00284	
100	54	0,00667	
101	27	0,00333	
110	12	0,00148	
111	190	0,02347	
1000	165	0,02038	
1001	18	0,00222	
1010	41	0,00506	
1011	198	0,02446	
1100	12	0,00148	
1101	190	0,02347	
1110	198	0,02446	
1111	3111	0,38426	0,449436579
Test 2 noise = 14.50%			

i	x_i	p_i	$p_i \setminus \{noise\}$
0	3661	0,45219	0,540847983
1	47	0,00581	
10	50	0,00618	
11	43	0,00531	
100	39	0,00482	
101	19	0,00235	
110	31	0,00383	
111	201	0,02483	
1000	172	0,02125	
1001	16	0,00198	
1010	61	0,00753	
1011	269	0,03323	
1100	15	0,00185	
1101	174	0,02149	
1110	190	0,02347	
1111	3108	0,38389	0,459152017
Test 3 noise = 16.39%			

i	x_i	p_i	$p_i \setminus \{noise\}$
0	3645	0,45022	0,540720961
1	45	0,00556	
10	62	0,00766	
11	33	0,00408	
100	59	0,00729	
101	19	0,00235	
110	37	0,00457	
111	227	0,02804	
1000	172	0,02125	
1001	18	0,00222	
1010	58	0,00716	
1011	233	0,02878	
1100	18	0,00222	
1101	195	0,02409	
1110	179	0,02211	
1111	3096	0,38241	0,459279039
Test 4 noise = 16.74%			

i	x_i	p_i	$p_i \setminus \{noise\}$
0	3804	0,46986	0,554761558
1	44	0,00543	
10	26	0,00321	
11	29	0,00358	
100	46	0,00568	
101	22	0,00272	
110	25	0,00309	
111	225	0,02779	
1000	164	0,02026	
1001	13	0,00161	
1010	35	0,00432	
1011	209	0,02582	
1100	17	0,00210	
1101	190	0,02347	
1110	194	0,02396	
1111	3053	0,37709	0,445238442
Test 5 noise = 15.30%			

GHZ_5 :

i	x_i	p_i	$p_i \setminus \{noise\}$
0	3762	0,464673913	0,59393748
1	38	0,004693676	
10	7	0,000864625	
11	7	0,000864625	
100	56	0,006916996	
101	12	0,001482213	
110	10	0,001235178	
111	64	0,007905138	
1000	91	0,011240119	
1001	2	0,000247036	
1010	4	0,000494071	
1011	39	0,004817194	
1100	4	0,000494071	
1101	30	0,003705534	
1110	37	0,004570158	
1111	437	0,053977273	
10000	105	0,012969368	
10001	3	0,000370553	
10010	3	0,000370553	
10011	5	0,000617589	
10100	4	0,000494071	
10101	9	0,00111166	
10110	8	0,000988142	
10111	126	0,015563241	
11000	57	0,007040514	
11001	14	0,001729249	
11010	22	0,002717391	
11011	183	0,022603755	
11100	10	0,001235178	
11101	169	0,020874506	
11110	206	0,025444664	
11111	2572	0,053977273	0,40606252
Test 1 noise = 21.76%			

i	x_i	p_i	$p_i \setminus \{noise\}$
0	3810	0,470602767	0,602371542
1	50	0,006175889	
10	12	0,001482213	
11	4	0,000494071	
100	55	0,006793478	
101	19	0,002346838	
110	1	0,000123518	
111	49	0,006052372	
1000	75	0,009263834	
1001	1	0,000123518	
1010	7	0,000864625	
1011	21	0,002593874	
1100	5	0,000617589	
1101	34	0,004199605	
1110	38	0,004693676	
1111	452	0,05583004	
10000	78	0,009634387	
10001	3	0,000370553	
10010	1	0,000123518	
10011	7	0,000864625	
10100	2	0,000247036	
10101	15	0,001852767	
10110	7	0,000864625	
10111	150	0,018527668	
11000	55	0,006793478	
11001	9	0,00111166	
11010	20	0,002470356	
11011	198	0,024456522	
11100	11	0,001358696	
11101	184	0,022727273	
11110	208	0,0256917	
11111	2515	0,05583004	0,397628458
Test 2 noise = 21.88%			

i	x_i	p_i	$p_i \setminus \{noise\}$
0	3763	0,464797	0,595882819
1	57	0,007041	
10	14	0,001729	
11	5	0,000618	
100	47	0,005805	
101	14	0,001729	
110	6	0,000741	
111	59	0,007288	
1000	102	0,012599	
1001	7	0,000865	
1010	5	0,000618	
1011	37	0,004570	
1100	4	0,000494	
1101	26	0,003211	
1110	28	0,003458	
1111	446	0,055089	
10000	100	0,012352	
10001	4	0,000494	
10010	2	0,000247	
10011	8	0,000988	
10100	4	0,000494	
10101	15	0,001853	
10110	9	0,001112	
10111	138	0,017045	
11000	68	0,008399	
11001	13	0,001606	
11010	29	0,003582	
11011	170	0,020998	
11100	15	0,001853	
11101	170	0,020998	
11110	179	0,022110	
11111	2552	0,315217	0,404117181
Test 3 noise = 22.00%			

i	x_i	p_i	$p_i \setminus \{noise\}$
0	3730	0,460721	0,584547877
1	45	0,005558	
10	13	0,001606	
11	3	0,000371	
100	47	0,005805	
101	19	0,002347	
110	8	0,000988	
111	47	0,005805	
1000	98	0,012105	
1001	7	0,000865	
1010	5	0,000618	
1011	25	0,003088	
1100	3	0,000371	
1101	26	0,003211	
1110	33	0,004076	
1111	428	0,052866	
10000	81	0,010005	
10001	1	0,000124	
10010	3	0,000371	
10011	15	0,001853	
10100	1	0,000124	
10101	10	0,001235	
10110	7	0,000865	
10111	158	0,019516	
11000	76	0,009387	
11001	9	0,001112	
11010	16	0,001976	
11011	192	0,023715	
11100	7	0,000865	
11101	163	0,020133	
11110	169	0,020875	
11111	2651	0,052866	0,415452123
Test 4 noise = 21.18%			

i	x_i	p_i	$p_i \setminus \{noise\}$
0	3785	0,467514822	0,603957236
1	38	0,004693676	
10	13	0,001605731	
11	5	0,000617589	
100	46	0,005681818	
101	18	0,00222332	
110	5	0,000617589	
111	46	0,005681818	
1000	112	0,013833992	
1001	3	0,000370553	
1010	6	0,000741107	
1011	39	0,004817194	
1100	6	0,000741107	
1101	33	0,004076087	
1110	37	0,004570158	
1111	442	0,054594862	
10000	92	0,011363636	
10001	0	0	
10010	4	0,000494071	
10011	14	0,001729249	
10100	1	0,000123518	
10101	7	0,000864625	
10110	11	0,001358696	
10111	162	0,020009881	
11000	62	0,007658103	
11001	19	0,002346838	
11010	32	0,003952569	
11011	200	0,024703557	
11100	15	0,001852767	
11101	165	0,020380435	
11110	196	0,024209486	
11111	2482	0,054594862	0,396042764
Test 5 noise = 22.59%			

Bibliography

[1] Frederic HOLWECK. *Introduction to Quantum Algorithms*.

[2] IBMQ. *IMB quantum cloud service*. URL: https://quantum-computing.ibm.com/.

[3] ID'quantique. URL: https://www.idquantique.com/.

[4] Apprendre en ligne. *KHI deux*. URL: https://www.apprendre-en-ligne.net/random/khideux.html.

[5] Quiskit. *Learn Quantum Computation using Qiskit*. URL: https://qiskit.org/textbook/preface.html.

[6] Wikipedia. *Chi-squared test*. URL: https://en.wikipedia.org/wiki/Chi-squared_test.

[7] Wikipedia. *Random Number Generation*. URL: https://en.wikipedia.org/wiki/Random_number_generation.

[8] Wikipedia. *Xorshift*. URL: https://en.wikipedia.org/wiki/Xorshift.