Problem Set No. 6

UBC Metro Vancouver Physics Circle 2018-2019

March 14, 2019

1 Quantum xerox machines

Just like a coin, a *classical bit* can be in precisely one of two states. Instead of heads and tails, we label these states with binary numbers 0 and 1.[1] A *quantum bit* is a sort of quantum coin, which like Schrödinger's dead-and-alive cat, can be in a mixture or *superposition* of the states 0 and 1:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle.$$

Here α, β are numbers we won't say much more about. The state of the coin $|\psi\rangle$ is a *vector* in two dimensions, but instead of unit vectors $\hat{\mathbf{i}}$ and $\hat{\mathbf{j}}$ in the x and y direction, we have unit vectors $|0\rangle$ and $|1\rangle$.

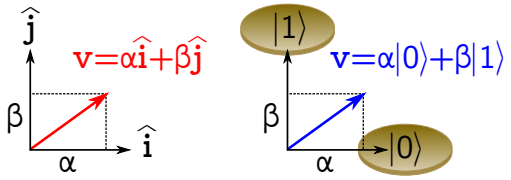


Figure 1: *Left.* A vector in the plane with two components. *Right.* State of a quantum coin.

There is a similar story for *multiple* quantum coins. Instead of a superposition over the two outcomes 0 and 1 of flipping a single coin, the state of n quantum coins is a superposition of the 2^n possible outcomes of flipping n classical coins. For instance, for $n = 2$ coins, the state is a superposition of $2^2 = 4$ possible outcomes:

$$|\psi\rangle = \alpha_{00}|0\rangle|0\rangle + \alpha_{01}|0\rangle|1\rangle + \alpha_{10}|1\rangle|0\rangle + \alpha_{11}|1\rangle|1\rangle.$$

Note that we can write unit vectors like $|0\rangle|1\rangle$ by simply "multiplying" the single coin unit vectors $|0\rangle$ and $|1\rangle$.[2] Since there are two single-coin unit vectors, for n coins we have $2 \times 2 \cdots \times 2 = 2^n$ unit vectors built this way, corresponding to the 2^n classical outcomes.

A *physical operator* A is something which changes the state, such as measuring the coin or flipping its orientation. But one of the basic requirements of quantum mechanics is that *operators are linear*:

$$A(c_1|\psi_1\rangle + c_2|\psi_2\rangle) = c_1 A|\psi_1\rangle + c_2 A|\psi_2\rangle.$$

We picture linearity in Fig. 3. This is a basic property of quantum mechanics: the result of physical

[1] In fact, "bit" stands for "binary digit".

[2] This is not the usual product of numbers, but a fancier operation called the *tensor product*.

Figure 2: Possible outcomes for flipping one, two or n classical coins.

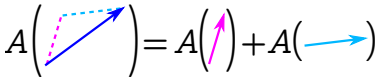


Figure 3: A visual representation of a linear operator acting on a sum of vectors.

operations on any state can be reduced to the result of operations on some nice set of unit vectors. For instance, if I want to implement a quantum operator which pokes Schrödinger's cat in any state, I just need to know how to poke the dead and the living cat separately!

We'll use the requirement of linearity to show that you cannot *clone* a quantum coin. There is no operator U acting as follows:

$$U|1\rangle|\psi\rangle = |\psi\rangle|\psi\rangle,$$

i.e. copying an arbitrary state $|\psi\rangle$ of a single quantum coin (in the second position) onto a coin freshly prepared in some known state, e.g. $|1\rangle$ (in the first position). This is very different from classical bits, where you can flip a coin and I can write down the result, effectively cloning the information. We will assume that U exists, and prove it must be non-linear. Taking linearity as a basic tenet of quantum mechanics, it follows that U cannot exist. Put a different way, there are no xerox machines for copying a quantum state!

1. Pick a particular state
$$|\psi\rangle = |+\rangle.$$
Since we're assuming U can clone any state, this is fine! Using the definition of U, evaluate
$$U|1\rangle|+\rangle.$$

2. Suppose that $|+\rangle$ is the specific state
$$|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle).$$
Use the linearity of U, and *then* the xerox property. What do you get now?

3. Argue that the answers from parts 1 and 2 are inconsistent. Hence, the quantum xerox machine U cannot exist!

4. Generalise your result to show that a state of n coins cannot be cloned.

5. A *teleportation operator* T does not clone a quantum coin, but shifts it somewhere else:
$$T|1\rangle|\psi\rangle = |\psi\rangle|1\rangle.$$
For instance, Alice could hold the first coin, and Bob the second; they have a teleportation device which moves the state of Bob's coin onto Alice's. Is this operator linear?

2 Hamming it up

An *error-correcting code* is a way to store information that protects against physical corruption of the storage medium. For example, suppose I have some classical information like a single bit b which can be 0 or 1, and some probability that the thing I am storing this bit on might get lost or erased. What should I do? Well, the simplest thing I could do is copy the bit a few times:

$$0 \to 00$$
$$1 \to 11.$$

Then, if I lose any one bit, I can just look at the other one. The information we'd like to store is called the *logical state*, while the encoded, larger number of bits is called the *physical state*.

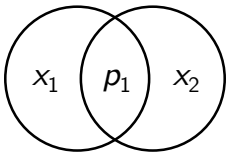
1. Design an error-correcting code that stores two bits worth of logical information in three physical bits. That is, fill in the blanks in

$$00 \to \_\ \_\ \_ \tag{1}$$
$$01 \to \_\ \_\ \_ \tag{2}$$
$$10 \to \_\ \_\ \_ \tag{3}$$
$$11 \to \_\ \_\ \_ \tag{4}$$

   so that your logical bits are protected against the erasure of one physical bit.

Now we study a somewhat more interesting error-correcting code called the *Hamming code*. It will let us store 4 logical bits into 7 physical bits.

Let's first revisit part 1. Although we can solve it using trial and error, there is a slicker approach. To see this, draw a Venn diagram with two circles. Call the two logical bits x_1 and x_2 and put them into the center of the two circles.



Then calculate the third bit that sits in the overlap of the two circles, according to $x_1 \oplus x_2 = p_1$, where $\oplus$ means you should follow the rule

$$0 \oplus 0 = 0$$
$$0 \oplus 1 = 1$$
$$1 \oplus 0 = 1$$
$$1 \oplus 1 = 0$$

Notice that $1 \oplus 1 = 0$, not 1! This is called addition "mod" 2, because everytime your sum gets up to 2 you go back to zero.[3]

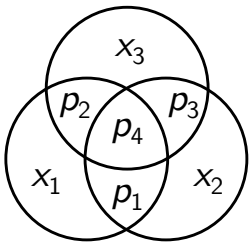
[3] If you know how to read a clock, you already know how to add mod 12! For instance, 3 hours after 11 is 2.

2. Use x_1, x_2, p_1 as your three physical bits and fill in the table above (this might be different than how you filled it in before).

3. Using your Venn diagram, how can you fill in the x_1 bit if you know p_1 and x_2? How about x_2 from p_1 and x_1?

Since it's possible to recover x_1 and x_2 when one bit is erased from the Venn diagram, this method gives an error-correcting code storing two logical bits into three physical ones, and corrects for the erasure of one bit. Notice that we could also use our Venn diagram backwards: we could call p_1 the logical bit. Then we notice that erasing any one of x_1, x_2, p_1 still allows us to construct p_1 again. So we can also view this as an error correcting code storing one logical bit into three physical ones! Let's generalise this construction to get the Hamming code.

4. First, we add another circle to our Venn diagram and relabel the regions as follows:



   You can use the $\oplus$ rule to store the 3 logical bits x_1, x_2, x_3 into 7 physical bits (xs and ps). How can we determine p_4 from the xs? Convince yourself that, using the Venn diagram, you can erase any bit and still recover x_1, x_2, x_3.

5. Now use your Venn diagram with three circles backwards: convince yourself that we can construct any p even after any one bit has been erased. This means you can actually store 4 logical bits into 7 physical ones!

3 Summoning: possibilities and impossibilities

This problem is part of Alex May's talk "Early days with quantum tasks". See the lecture notes for further background and context.

Consider the summoning task shown in Fig. 4. Alice receives some quantum information at the yellow dot. At the black *call points* c_1, c_2 she receives classical bits b_1, b_2. If $b_i = 1$, for either $i = 1$ or $i = 2$, she must return her quantum information to the corresponding blue *return point* r_i.

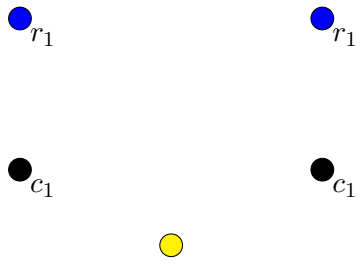


Figure 4: An impossible summoning task.

Notice that the return point r_1 is outside of the light cone of the call point c_2, and the return point r_2 is outside of the future light cone of the call point c_1. In this problem, we will argue that it is impossible to complete this task with a perfect success rate. Our strategy will be to assume it is possible, and show our assumption results in nonsense. We can then conclude that our assumption was wrong and the task is impossible.

1. What classical information does Alice have available to her at r_1? What about at r_2?

2. Usually, Alice gets a call at c_1 OR at c_2. Imagine though that we played a trick on her, and asked for the state back at both c_1 AND c_2. If Alice is running her perfect protocol that always completes the task, what happens when we play our trick?

3. Use the results of a previous exercise or something you've learned in these notes to conclude this is nonsense, and so it must be that the task is impossible.

We're now going to change gears, and consider a cyclic task with three diamonds, depicted in Fig. 5. The idea will be to carry it out with a $((2, 3))$ error-correcting code. Suppose Alice receives the state at s and promptly turns it into three shares in a $((2, 3))$ error-correcting code. Now she needs to decide where to send each share. Should she send one share to each diamond, or two shares to one diamond?

4. Notice that each diamond is the same as every other: it connects to one diamond and is connected to from one other diamond. Use this fact to argue that Alice should split into three parties, and send one share to each diamond.

5. Now Alice has brought one share to each diamond. At the c_i each party receives the information b_i, and uses this information to decide where to send their share.There are two cases: $b = 0$ and $b = 1$. Decide where to send the share in the error-correcting code in each case, and argue this always completes the summoning task.

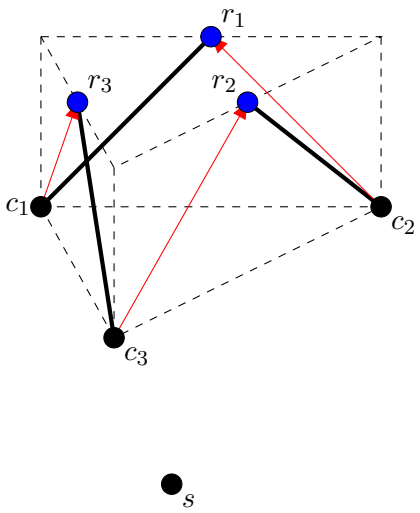


Figure 5: A cyclic task, initially (and incorrectly) thought to be impossible.

6. **Bonus.** What happens in the protocol if two of the bs are 1? If you know for sure you will get exactly two bs with $b = 1$, is there a protocol that will always result in the state being handed over at one of the diamonds where $b = 1$? What if you're not sure how many call points will give $b = 1$?