# Quantum Programming Foundations: History and Overview

Jens Palsberg

Apr 1, 2019

## Outline

**Hook:** Quantum computers are here! We have algorithms for them but how do we actually run a quantum algorithm on a quantum computer? To answer this question, we need a sense of what the algorithms are like, how to program them in a programming language, and how to run a program.

**Purpose:** Persuade you that you can learn quantum programming.

**Preview:**

1. We will take the time to go through the details.
2. The quantum programming languages look familiar.
3. The math is linear algebra, probabilities, and complex numbers.

**Transition to Body:** First let me tell you about the pace of the course.

**Main Point 1:** We will take the time to go through the details.
[April is about foundations and algorithms, ending with a midterm exam]
[May is about a handful of quantum programming languages]
[June is about comparison of languages and about research questions]

**Transition to MP2:** How weird are the quantum languages?

**Main Point 2:** The quantum programming languages look familiar.
[Some of the quantum languages are Python libraries]
[Some of them are entirely new designs, yet have familiar aspects]
[Some of the languages model the connections between the qubits]

**Transition to MP3:** The math that we need is stuff that many other courses use as well.

**Main Point 3:** The math is linear algebra, probabilities, and complex numbers.
[Double-slit and probabilities as complex numbers]
[Computing with vectors of complex numbers]
[Two levels of probabilities]

**Transition to Close:** So there you have it.

**Review:** We will go slow, we have a real chance of learning the quantum languages, and the math is mostly something that many people learn as undergraduates.

**Strong finish:** We have a dream team to do the teaching effort. In addition to me, we have Auguste Hirth as the teaching assistant; he is working tirelessly on getting quantum programs to run. We also have Thomas Jerry Chang as the reader; he will grade the homework and give you all the feedback you need to succeed.

**Call to action:** Get going on the first homeworks today! We are ramping up fast, both on the math and on the programming.

## Detailed presentation

**Hook:** Quantum computers are here! Many companies are working on quantum computing, more and more researchers work on quantum computing, and this year the U.S. government started the National Quantum Initiative that allocated more than $1 billion to research in quantum computing. We have algorithms for quantum computers but how do we actually run a quantum algorithm on a quantum computer? To answer this question, we need a sense of what the algorithms are like, how to program them in a programming language, and how to run a program. This course is the first course in the world that will teach all that, including programming in multiple quantum programming languages.

**Purpose:** Persuade you that you can learn quantum programming.

**Preview:**

1. We will take the time to go through the details.
2. The quantum programming languages look familiar.
3. The math is linear algebra, probabilities, and complex numbers.

**Transition to Body:** First let me tell you about the pace of the course.

**Main Point 1:** We will take the time to go through the details.

[April is about foundations and algorithms, ending with a midterm exam]
The goal for April is to get a detailed understanding of a handful of algorithms that we will run in May. Those algorithms are short, if we count lines of code, yet also complex, if we count the brain cycles needed for a newcomer to understand them. The way we will get there is to begin with two weeks on foundations and then continue with two weeks on algorithms. In the two weeks about foundations, I will cover the math needed to understand what quantum computing is and how the algorithms work. Let me get into this a little bit already now.

Quantum computing may fundamentally change what is efficiently computable. How? The idea is to scale computation exponentially with the size of the computer. This may allow us to solve otherwise intractable problems in optimization, chemistry, and machine learning. Those three areas are indeed the most promising application areas for quantum computing. Among those three areas, the front runner is optimization. We have all heard of Shor's algorithm for factoring numbers, but actually using it to factor the numbers we use in crypto is way into the future. Much more realistic for near-term quantum computers is a quantum algorithm for finding approximate solutions to optimization problems. Those problems can be NP-complete problems like graph coloring, traveling salesman, and integer linear programming. A paper from MIT in 2014 by Farhi and his colleagues presented such an algorithm and it generated a ton of interest. DARPA is starting a program devoted entirely to this algorithm and it descendants. If we can use quantum computers to quickly get good approximations to large NP-complete problems, it will be a game changer.

Where are we with building quantum computers? Google has announced their 72-qubit superconducting machine, IBM is testing a machine with 50 quantum bits (qubits) also using superconducting technology, and Innsbruck has 20-qubit machine based on trapped-ion technology. Machines with 100 qubits are around the corner and 1,000 qubits seem within reach.

How many qubits can we simulate on a classical computer? Recently, NASA simulated 70 qubits on its supercomputer, and I have seen reports of academics who have simulated 49 qubits.

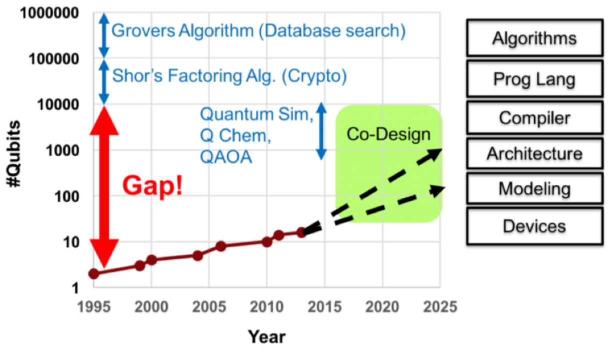
The key here is that the challenge doubles for every additional qubit and nobody believes we can go beyond simulation of 100 qubits. So we are on the cusp of a quantum advantage.

Current quantum computers have high error rates, or, as John Preskill at Caltech has said: they are noisy. He called them NISQ computers, short for Noisy Intermediate-Scale Quantum computers. The errors will limit the potential until better error correction kicks in.

Let's compare two computers: one classical and one quantum.

The classical computer in our comparison is Intel Xeon Phi Processor, also known as "Knight's Corner". It is a commonly used processor in supercomputing clusters. This processor has a clock frequency of 1 GHz. In 2017, researchers measured the "soft error rate", which are radiation-induced errors that can flip states. The got the soft error rate to be 1 error per 1 thousand years.

In contrast, the IBM Q16 Rueschlikon is a quantum computer. It clocks at 5.26 GHz. The statistics are obtained on the IBM calibration data page in October 2018. Its single-qubit gate error is 2 errors for every 1,000 operations, its multi-qubit gate error 44 errors for every 1,000 operations, and its read out error is 6 errors for every 100 measurements. All of those error rates are orders of magnitude more frequent than the error rate of the Knight's Corner.



Two of famous quantum algorithms are Shor's algorithm and Grover's algorithm. Shor's algorithm factors integers, which is useful for breaking crypto, while Grover's algorithm searches an unstructured database super quickly. As it happens, those algorithms require more than 10,000 qubits to do anything useful. Optimizing compilers may have a role to play here. Once we look closer at the optimization problem, it looks a lot like a design automation problem of the kind known from FPGA design and the like.

How good is a quantum computer? The key concept is the *quantum volume*, or the space-time product:

$$\text{quantum volume} \;=\; \#\text{qubits} \times \#\text{operations}$$

However, this must be tempered with the error rate of each operation. As we saw before, an operation on a couple of qubits has a much higher error rate than an operation on a single qubit. How far can we go in the near future? People are hoping for:

$$\text{near-future quantum volume} \;=\; 100 \text{ qubits} \times 1{,}000 \text{ operations}$$

The quantum computer stops working when decoherence kicks in, that is, when the quantumness goes away. Today, this happens within a second. Quantum computers today are what classical computers were in the 1950s. As an analogy that I learned from Alan Ho at Google, Google's current quantum computer has 72 qubits, and it just so happens that the Harvard Mark 1 computer from 1944 could store 72 numbers. By the way, one of the first programs on the Mark 1 was run by John von Neumann who was working on the Manhattan Project and needed some computation done. So, 72 qubits in 2019 and 72 numbers in 1944. We have some way to go before quantum computing becomes practical.

Let me compare with Moore's law. Moore's law says that the number of transistors on a square-inch die doubles every two years. We can translate this to say that the number of bits available for computing doubles every two years. Intuitively, if we have $2^{100}$ bits, we have a state space with $2^{100}$ dimensions. Let us contrast this with quantum computing. In quantum computing, every additional qubit doubles the number of dimensions of the state space. So, if we have 100 qubits, we have a state space with $2^{100}$ dimensions. Now I am waiting for the quantum equivalent of Moore's law that will talk about that the number of qubits increases by one every so often. How often? At the moment, my impression is that the number of qubits increases by one every month. This is exciting!

Alan Ho at Google talks about Neven's law, after Hartmut Neven. At Google, the reliability of gates is improving linearly over time. Neven observed that as you improve the reliability of the 2-qubit gates, you can linearly increase both the entanglement and the depth of the circuit. Because the equivalent classical computational power is exponential in both number of qubits and gate depth, this leads to a double exponential in growth. At the moment, the most reliable qubit in the world is from UCLA, from Eric Hudson's group in Physics, at 99.99 percent reliability.

What kind of algorithm is a good candidate for running on a quantum computer?

1. Small input, lots of computation, small output. Example: Shor's algorithm (input = an integer; output = two integers).

2. A result that we can verify easily. Example: Shor's algorithm (multiply the integers).

3. A subroutine for a classical computation. Example: Simon's algorithm (the quantum computer outputs equations, while the classical computer solves the equations).

We are going to spend April on the foundations and on some of the algorithms of quantum computing. CCLE has a ton of reading material for April. Additionally, we have a course reader for sale for \$55, which has material for April as well. I wish I could give you less material to read, yet I have found that no single source covers everything well. The material on CCLE stems from many different professors around the world, and every single piece has something interesting that is unique to that piece. However, those materials also have overlap. I saw myself as having two options: either wait a year with offering the course until I have sorted all this out and written my own material, or offer the course now and give material with overlap. I decided to go for it!

CCLE also has a ton of homework for April. Some of the homework is on the foundations, which means math that boils down to mostly linear algebra and Boolean functions. I give you this homework for two reasons. One is to get you to page in some linear algebra that you already know and to learn some other linear algebra that you don't know. The other reason is to think through some points that are directly relevant to the algorithms that we will cover later in April.

Some other homework in April is programming homework. Some of it is about running quantum circuits, to try out some ideas in the simplest possible setting. Some other programming homework is about getting into the details of the problems that the quantum algorithms can solve. You will do that by programming solutions on classical computers. This means that once we get to the quantum algorithms and the problems they solve, you have already solved each of them on a classical computer. You can use any language you like for those homeworks.

Overall, April has homework due twice a week, for the purpose of getting you ready for programming quantum computers in May. You will do the homework in April individually, and we will have a midterm exam in early May. We will also have five online quizzes in April.

[May is about a handful of quantum programming languages]

Every company that is building quantum computers also has its own quantum programming language. Google has Cirq, IBM has Qiskit, Microsoft has Q#, and Rigetti has PyQuil. Each of those companies has built approximately one quantum computer, so we can say that in the quantum world, we have one language per computer. No standardization on that front! However, hopefully this will change over time, and the goal should be the same as for classical computing: a language should run on many different computers. In addition to languages from companies, we also have languages from universities, another handful of them.

Which language will emerge as the winner? This is way to early to say. My guess is: none of them; people will design better languages. But we should try to grasp what language designers care about, how language designers envision language support for quantum algorithms, and how the ideas compare. We will get into all that in May by programming and running the algorithms that we learned in April. I am trying to get licenses for all of us to run on quantum computers with cloud access, and otherwise we will run on quantum simulators.

The homework in May will cover three of the quantum languages. This will give everybody three kinds of experience. First, we will find out how easy or difficult each language is to work with. Second, we will find how similar or different an algorithm comes out looking in different languages. Third, we will find out how well each language supports abstractions that will help scale to algorithms that work with 1,000 qubits. CCLE has the homework for May already, though we are working on fine-tuning it to give everybody the best learning experience.

In May you will work in groups. The maximum group size is three. We have 72 people in the course so if every group has three people, we will have 24 groups. Each group will submit quantum programs throughout May, spilling into June. You will form the groups, the deadline for group formation is Apr 30, 2019. Submit the group member names and the group name on CCLE.

[June is about comparison of languages and about research questions]

In June, I will wrap up the course by getting into a comparison of the languages, and by outlining a suite of research questions. I will talk about research questions to give you a sense of my perspective on where the field is going, which obstables have to be overcome, and which problems are ripe for being solved in the next couple of years. Some of you may be interested in getting involved in some of that research yourself; if so, please come and talk with me.

We will have final group project in June. The course has no final exam.

My plan for grading is straightforward. This is a graduate course and I would much prefer to give only A, B, C grades. If I sense that you really tried, even if you have lots of stuff wrong, I will give you either an A or a B, I hope that this will be true for everybody. My plan is to limit the A's to 45 percent, so my ideal case is that 45 percent of you gets an A and that 55 percent gets a B. However, I reserve the right to give a C or even below if I see little evidence of effort.

Let me make a shoutout to the undergraduate students in the course. I am delighted that fourteen of you are here and I want to encourage you to speak up. I know that you are outnumbered by the graduate students, yet please contribute to the discussions as much as you can.

Surely you have noticed that I have no slides. This is how I am going to roll the entire quarter. I am going to use the blackboard. The reason is that I want to force myself to slow down. Hopefully the slow pace will give you time to think and to ask questions. I hope you will ask a ton of questions.

We will use piazza in the course and I am going to give ten percent of your grade based on participation on piazza. Some examples of participation: ask a question, answer a question, and send a link to a newspaper article and comment on it. Some other examples could be to list a typo in the CCLE material and suggest a fix, show a work-through of a new example, and link to a research paper and comment on it. Plan on posting at least five times on piazza under your own name this quarter. The goal is to build a community around quantum computing at UCLA.

**Transition to MP2:** How weird are the quantum languages?

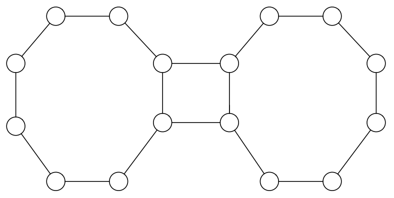
**Main Point 2:** The quantum programming languages look familiar.

[Some of the quantum languages are Python libraries]
Cirq from Google and PyQuil from Rigetti are both Python libraries. This means that we can write Python code that runs on a classical computer and makes calls to a quantum computer. This is convenient because a lot of the code we need to write is not quantum at all. Rather, much of the code is for set up, post-processing, and input-output, which all can be written in Python. So, Python is the host language and all the quantum stuff is done by calling a quantum library.

[Some of them are entirely new designs, yet have familiar aspects]
Q# is a domain-specific language that was designed entirely for the purpose of expressing quantum algorithms. Program execution proceeds in a manner similar to the Python libraries: a classical computer controls the computation and makes calls to a quantum computer. However, in Q# the boundary is more fluid and up to the compiler.

[Some of the languages model the connections between the qubits]
For classical computers, while many companies make them, really they are all the same in one important respect: they are all based on CMOS. For quantum computers, this is not at all the case. Different companies build qubits in radically different ways. This affects quantum computing in particularly one important way: what if we want to do an operation on two qubits; are they connected? On some quantum computers, any operation on two qubits requires that they are connected. On other quantum computers, an operation on two qubits can work on any two qubits.

Rigetti has a quantum computer with 16 qubits that are arranged as two octagons with two connections that cut across.



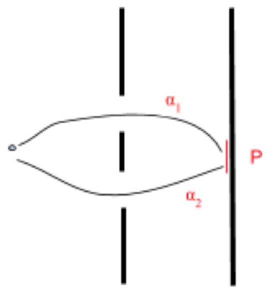
Languages that model the connections give programmers a better chance of writing efficient programs. However, this is also a low-level style of programming that requires a programmer to pay attention to aspects that we would love to abstract away. Perhaps compilers can help. The problem of mapping the qubits in a program to the qubits on a specific quantum computer is akin to problems in design automation. Specifically, mapping problems in FPGA design and ASIC design remind of the mapping problem for quantum computing.

**Transition to MP3:** The math that we need is stuff that many other courses use as well.

**Main Point 3:** The math is linear algebra, probabilities, and complex numbers.

[Double-slit and probabilities as complex numbers]

Feynman said that the essence of quantum mechanics is the Double Slit Experiment. This experiment goes back to Thomas Young in 1801.



The apparatus consists of a source of light, a wall with two narrow slits, and a viewing screen. In the double-slit experiment, we shoot photons one at a time toward the wall with the two slits. Where each photon lands on a second wall is probabilistic. If we plot where photons appear on the back wall, some places are highly likely, some unlikely.

So far this is straightforward: we might justify this behavior by a theory where each photon has a degree of freedom that determines which way it goes. What is weird is the following. For some interval on the second wall, let us define three probabilities:

- let $P$ be the probability that the photon lands in the interval with both slits open;
- let $P_1$ be the probability that the photon lands in the interval if only slit 1 is open; and
- let $P_2$ be the probability that the photon lands in the interval if only slit 2 is open.

We would think that $P = P_1 + P_2$. But experiments show that this is false! Even places that are never hit when both slits are open, can sometimes be hit if only one slit is open.

Slogan: God plays dice, but they aren't normal dice.

The way to make this work is to change probabilities from real numbers to complex numbers. The people in quantum mechanics use the word *amplitudes* instead of probabilities.

The central idea of quantum mechanics is that to fully describe a system, you need a probability (an amplitude) for each possible configuration.

The Born rule: if the amplitude of a configuration is $\alpha$, then the probability $P$ of seeing that configuration as an outcome is

$$\begin{aligned} P &= |\alpha|^2 \\ &= (\text{the real part of } \alpha)^2 + (\text{the imaginary part of } \alpha)^2 \end{aligned}$$

Now we can redo the Double Slit Experiment with amplitudes instead of probabilities.

- let $\alpha$ be the amplitude that the photon lands in the interval with both slits open;
- let $\alpha_1$ be the amplitude that the photon lands in the interval if only slit 1 is open; and
- let $\alpha_2$ be the amplitude that the photon lands in the interval if only slit 2 is open.

We do get $\alpha = \alpha_1 + \alpha_2$, and now also get (using $\alpha_1^*$ to denote the conjugate of $\alpha_1$):

$$P = |\alpha|^2 = |\alpha_1 + \alpha_2|^2 = |\alpha_1|^2 + |\alpha_2|^2 + \alpha_1^*\alpha_2 + \alpha_1\alpha_2^*$$

Example: suppose $\alpha_1 = \frac{1}{2}$ and $\alpha_2 = -\frac{1}{2}$. By the Born rule, the probability that the photon lands in the interval with one bit slit open is $\alpha_1^2 = \alpha_2^2 = \frac{1}{4}$. But if both slits are open

$$P \;=\; |\alpha_1|^2 + |\alpha_2|^2 + \alpha_1^* \alpha_2 + \alpha_1 \alpha_2^* \;=\; \frac{1}{4} + \frac{1}{4} - \frac{1}{4} - \frac{1}{4} \;=\; 0$$

So, in quantum mechanics, probabilities (amplitudes!) can cancel each other out.

Now let us ask "how does a particle that went through the first slit know that the other slit is open?" In quantum mechanics, this question is ill-formed. Particles don't have trajectories, but rather take all paths simultaneously, in superposition. This is where we get the power of quantum computing.

[Computing with vectors of complex numbers]

The idea of quantum computing is that those complex numbers from quantum mechanics are the building blocks. Classical computing is about computing with bits, while quantum computing is about computing with complex numbers. Just like the state of a classical computer is given by a vector of bits, the state of a quantum computer is given by a vector of complex numbers. What can we do with those complex numbers? I will get into this in the next lecture.

[Two levels of probabilities]

Quantum computing has two levels of probabilities. Probabilities on top of probabilities. At the bottom level we have probabilities from quantum mechanics. At the top level we have probabilities of the kind that we find in statements like: this algorithm gives the correct answer with high probability. The top layer turns out to be what quantum algorithms are good at: finding approximate answers really fast. In contrast, quantum computers cannot get exact answers to hard problems much faster than a classical computer. For example, a quantum computer cannot find exact solutions to NP-complete problems in polynomial time. However, a quantum computer may be much better than a classical computer at finding approximate solutions to NP-complete problems.

**Transition to Close:** So there you have it.

**Review:** We will go slow, we have a real chance of learning the quantum languages, and the math is mostly something that many people learn as undergraduates.

**Strong finish:** We have a dream team to do the teaching effort. In addition to me, we have Auguste Hirth as the teaching assistant; he is working tirelessly on getting quantum programs to run. We also have Thomas Chang as the reader; he will grade the homework and give you all the feedback you need to succeed.

Let me introduce the Quantum Computing Drinking Game. In the company of friends and fellow students, you fire up a youtube video on quantum computing or you read lecture notes. Now you listen and watch for the following quotes.

Albert Einstein: "God does not play dice." (1926)
Richard Feynman: "Nature isn't classical, dammit." (1981)
Everybody: "$\frac{1}{\sqrt{2}}$" (every year)

Every time you hear or see one the quotes, you take a sip.

**Call to action:** Get going on the first homeworks today! We are ramping up fast, both on the math and on the programming.