Teaching quantum computing through quantum software

Olivia Di Matteo

CAP Congress, 8 June 2022



THE UNIVERSITY OF BRITISH COLUMBIA

Electrical and Computer Engineering Faculty of Applied Science

Quantum computing education

Traditionally, quantum computing courses have:

- Targeted primarily graduate students (often physicists)
- Focused mostly on the **underlying theory**
- Not actually taught how to *program* a quantum computer



Quantum computing education

The QC industry is growing. We need more courses that:

- Target undergraduates
- Target non-physicists
- Focus on actually writing software and algorithms for quantum computers, using industry-relevant tools



Teaching quantum computing through quantum software

Overview of CPEN 400Q: Gate-model quantum computing

- Course content
- Teaching demo +
- Assessment
- Exploring background-specific content design

CPEN 400Q

- ECE department at UBC, Jan-Apr 2022
- First dedicated undergraduate quantum computing course at UBC
- 31 undergrads, and 1 physics grad student
- First month on Zoom, then (mostly) in-person



Undergraduate Majors

- Computer Engineering
- Electrical Engineering
- Integrated Engineering
- Engineering Physics
- Computer Science / Physics

Slides & demos available open source: https://github.com/glassnotes/CPEN-400Q

CPEN 400Q

Course learning outcomes

Core goal: **learn how to program quantum computers** in a hands-on, software-focused setting.

- Describe the societal importance and implications of quantum computing
- Explain the theory and principles behind gate-model quantum computing
- Describe the operation of key quantum algorithms
- Implement basic and research-level quantum algorithms using Python and PennyLane

In this course you will implement everything you learn!

Slides & demos available open source: https://github.com/glassnotes/CPEN-400Q

Content: define and compute basic concepts

- Write quantum states, compute action of gates and results of measurements
- Define superposition and entanglement
- Describe which algorithms give quantum speedups and which don't
- Express quantum computations in the quantum circuit model
- Describe the operation and structure of key quantum algorithms...



Content: implement core algorithms in software

- Quantum teleportation
- Deutsch's algorithm
- Grover's search algorithm
- Quantum Fourier transform
- Quantum phase estimation
- Shor's algorithm
- Variational quantum classifier
- Variational quantum eigensolver
- Quantum approximate optimization algorithm
- Basic Hamiltonian simulation



Content: teaching methods

- Half annotation of slides on iPad, half live coding
- Lots of pictures
- Tried to leverage concepts and algorithms they were familiar with before showing analogous quantum algorithms



Content: the first weeks

- Students did not know quantum mechanics, but they *did* know how to program
- Manually programmed the components of simple quantum computation before jumping to quantum software
- All coding afterwards was using **PennyLane**

Demo 3: measurement def measure(state, shots=50): prob 0 = np.abs(state[0]) ** 2# prob 0 = state[0] * state[0].conj() prob 1 = np.abs(np.vdot(ket 1(), state)) ** 2 **return** np.random.choice([0, 1], size=shots, p=[prob 0, prob 1]) some state = apply multiple U([H, Z, X], ket O())measure(some state, shots=20) array([1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1]) def guantum algorithm(): state = superposition(1j/2, np.sqrt(3)/2) state = apply multiple U([Z, H, X, H, X, H], state) return measure(state) quantum algorithm() array([1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1])

Content: the first weeks

- PennyLane is an open-source quantum software framework; development led by startup Xanadu in Toronto
- Write and run quantum circuits just like Python functions!
- Teaching/learning through software allowed students to apply a familiar language and tools to new concepts

```
Demo 2: basis rotation
dev = gml.device('default.gubit', wires=1, shots=20)
def basis rotation():
    gml.Hadamard(wires=0)
    qml.S(wires=0)
@gml.gnode(dev)
def circuit(x, y, z):
    qml.RX(x, wires=0)
    qml.RY(y, wires=0)
    qml.RZ(z, wires=0)
    # Rotate back to computational basis
    qml.adjoint(basis rotation)()
    return gml.sample()
circuit(0.1, 0.2, 0.3)
```

Content: the first weeks

Teleportation is a great quantum algorithm with which to end the first few weeks:



Access to quantum software helps us express this in terms of smaller algorithmic building blocks that can be combined and reused.

Hands-on demo: quantum teleportation

To code along go to:

https://bit.ly/38PVFwO

Assessment

Three components for grading:

- Computational assignments (30%)
- Weekly quizzes (20%)
- Final project (50%)

(No exams!)

Assessment: assignments

- Consisted of solving programming problems
- Distributed and submitted through GitHub
- Grading scripts with test cases were usually provided
- Points for comments, formatting, and source/collaborator citation

Issues:

- Convoluted submission instructions
- Tricky to balance autograding and manual providing of feedback
- Tried to sneak in extra concepts: some liked this, some didn't.

Next time: GitHub classroom?

Assessment: quizzes

- Distributed and submitted through GitHub
- Individual, but can consult documentation and notes
- 4 hour time window to complete

... this didn't really work. Setup too convoluted for such a short time window.

Next time: in-class quizzes in pairs.

```
import pennylane as qml
 2
 3
   def quiz_1(x, y):
 4
       """Write and execute a QNode that implements the following circuit:
        0: --H----- cC----
7
       1: --RX(x)---|C---|
8
        2: -RY(y) - - | (X)
9
11
       Args:
           x (float): Angular parameter for X rotation
           v (float): Angular parameter for Y rotation.
14
       Returns:
           float: The analytical expectation value of X on the final qubit
```

float: The analytical expectation value of X on the final qubit obtained after executing your QNode.

YOUR CODE HERE

```
return
```

Assessment: final project

Replicate the results of a recent research paper in software.

Three (equally-weighted) components:

- Software implementation
- Class presentation (on Zoom) + live coding demo
- Companion report detailed their process and issues

Very challenging, but they did really well!

Next time: more checkpoints; permute order of course content...

The textbook

(codebook.xanadu.ai)

Xanadu Quantum Codebook:

- Free, self-paced, and hands-on resource that teaches quantum computing
- Target audience is software developers who know **Python**, and a little bit of **linear algebra**
- Readers **learn by doing** by solving programming exercises.

I.10 What did you expect?

Codercise I.10.1. Design and run a PennyLane circuit that performs the following, where $\langle Y \rangle$ indicates measurement of the <code>PauliY</code> observable.

$$|0\rangle - R_x(\pi/4) + H + Z - \bigwedge \langle Y \rangle$$

1	<pre>dev = qml.device('default.qubit', wires=1)</pre>
2	
3	@qml.qnode(dev)
4 *	<pre>def circuit():</pre>
5	#######################################
6	# YOUR CODE HERE #
7	#######################################
8	
9	# IMPLEMENT THE CIRCUIT IN THE PICTURE AND MEASURE PAULI Y
10	
11	return
12	
13	<pre>print(circuit())</pre>
14	

The textbook

< Back to Modules

H.1

H.4

H.5

H.6

H.7

(codebook.xanadu.ai)

×

Graph to navigate content

H.3 Hamiltonians

We've lumped some of the constants into $\alpha = eB/2m_e$ for convenience. The Z operator has eigenvalue +1 when the magnet is aligned, and -1 when anti-aligned, so the energy is indeed lower in the first case:



To get the unitary according to (1), we must exponentiate (2), which is simply a Z rotation:



Codercise H.3.1. (a) Complete the code to build the unitary (3). We can verify the output is unitary using unitary check.

2 """Creates a unitary operator to evolve the state of an electron in
3 a magnetic field.
4
5 - Args:
6 B (float): The strength of the field, assumed to point in the z direction
7 time (float): The time (t) we evolve the electron state for.
8
9 - Returns:
10 array[complex]: The unitary matrix implementing time evolution.
11
12 e = 1.6e-19
13 m e = 9.1e-31
14 alpha = B*e/(2*m e)
15 #####################
0/6 Complete
of a complete

H.3 Hamiltonians

 $\begin{aligned} &-\cos\left(\alpha t_{1}/\tau_{1}+\tau_{1}-\sin\left(\alpha t_{2}/\tau_{1}-\tau_{1}-\tau_{2}\right)\right) \\ &=\cos^{2}(\alpha t)-\sin^{2}(\alpha t) \\ &=\cos(2\alpha t). \end{aligned}$

Similar manipulations show $\langle Y\rangle=\sin(2\alpha t)$ and $\langle Z\rangle=0$, with the angle changing at a rate 2α . A stronger field won't align the spin any better, but it will rotate the spin vector faster! This is called Larmor precession:



In general, the spin vector S of the electron will simply rotate around the z-axis with angular velocity Be/m_e , as you can check in the next exercise. It may seem odd that the magnetic field doesn't push the spin into alignment, but it turns out that in quantum as in classical physics, a *uniform* magnetic field cannot cause tiny magnets to align. Iron filings only arrange themselves along field lines because the strength of the field at the top and bottom of the filing is different!

Exercise H.3.2. Suppose that $|\psi(0)\rangle$ has expectations

 $(\langle X \rangle_0, \langle Y \rangle_0, \langle Z \rangle_0) = (x, y, z).$

Show that $|\psi(t)
angle = U(t)|\psi(0)
angle$ has expectations

 $(\langle X \rangle_t, \langle Y \rangle_t, \langle Z \rangle_t) = (\cos(2\alpha t)x + \sin(2\alpha t)y, -\sin(2\alpha t)x + \cos(2\alpha t)y, z),$

where for an operator O, $\langle O \rangle_t = \langle \psi(t) | O | \psi(t) \rangle$. This means a qubit starting in any state will simply precess clockwise around the z-axis with angular rate of change 2α .

Solution.

We now have a nice simple example where the unitary evolution is connected to the physics. Time to consider something more interesting!

Autograded coding exercises Companion textbook content

×

(3)

Ordering the material

Codebook content is divided into modules in a non-linear way: choose a path based on your own interests and background.



Hamiltonian simulation

Ordering the material



Physics-focused background

CS-focused background

Ordering the material: CPEN 400Q



G H S Variational Algorithms VQC => VQE => QAOA

Ordering the material: CPEN 400Q



How to learn which order works the best?

- All students are different
- Varying class composition but still mostly CPEN students
- Small class size
- Long time horizon



Other content questions:

- What is the right amount of quantum mechanics to teach?
- When to introduce Hamiltonians?
- When/how to facilitate a discussion about ethics?

Takeaways

- Students *love* live coding (even if you make mistakes!)
- Autograded quizzes/assignments are powerful tools but must be wielded wisely
- The order of material must be chosen to suit student backgrounds, but not clear what is the best way to choose it



Thanks & acknowledgments

- To my TA, Gideon Uchehara, for reviewing lectures, testing assignments, and helping out the students
- To the Xanadu team, for providing computing resources for in-class demos, and for fostering a work environment that enabled development of the Codebook
- To all the CPEN 400Q students for their patience and their valuable feedback!