Assignment 1

***Due**: Wednesday, September 17th, 2025 @ 11:59PM*

CPSC 4470/5470 Introduction to Quantum Computing (Fall 2025)

1 Introduction

Welcome to the first assignment for CPSC 4470/5470 (Introduction to Quantum Computing). Before getting started, please take a moment to read the course syllabus and familiarize yourself with the homework policies, including on late days and collaborations. Importantly, if you discussed with anyone besides the course staff about the assignment, *please list their names* in your submission.

Getting Started.

This assignment covers the basics of quantum computation, focusing on the first two weeks of classes. It has two parts, a *written portion* (52%) and a *programming portion* (48%). The tasks that are marked by "($\star$ pts)" are optional. Typesetting your solutions to the written portion is not mandatory but highly encouraged. Some basic familiarity with Python and object-oriented programming is required to complete the programming portion of this assignment. To start,

- Create a folder for Assignment 1, e.g., `A1/`
- Download the starter files for this assignment to that folder from the course website:
  - `written.tex`
  - `A1.py`
  - `requirement_A1.py` (Do not modify)
- Write your solutions in `A1.py` (for programming tasks) and in `written.tex` or on paper (for written tasks).
- Debug and test your solution locally by running '`python3 A1.py`' on command line. This will check for any violation of the requirements and run correctness tests. Feel free to add more tests in `A1.py`. Do not hardcode your solutions for each public test case.

Please see instructor's Ed post for more information and tips for the assignment.

Submission.

Once you have completed and are ready to submit, upload two files to Gradescope (accessed through Canvas): `written.pdf` and `A1.py`. Gradescope will immediately show the results from running the requirement test and public test cases. If your file fails the `requirement_A1.py` check, a **0** score will be assigned.

After the deadline, your written solution will be graded manually by our course staff; your programming solution will be graded using our auto-grading script that contains private test cases.

WRITTEN PORTION
This portion of the assignment has a total of 52 *points.*

2 Complex numbers

"Life is complex – it has both the real and imaginary parts." – *Unknown.* In class, we saw that a complex number α has a form: $\alpha = a_1 + a_2 i$, where $a_1, a_2 \in \mathbb{R}$ and $i = \sqrt{-1}$. In the following tasks, we will learn to work with complex numbers.

Task 2.1 (9 pts)

Given a *non-zero* complex number $\alpha = a_1 + a_2 i$, compute the following expressions. Please write your answers by specifying the real and imaginary parts.

(a) $\frac{1}{\alpha}$

(b) $|\alpha|^2$

(c) α^2

Task 2.2 ($\star$ pts)

Given a *non-zero* complex number $\alpha = a_1 + a_2 i$, compute the following expressions. Please write your answers by specifying the real and imaginary parts.

(a) $e^{-i\alpha}$

(b) $\frac{1}{2i}(\alpha - \alpha^*)$

Task 2.3 (6 pts)

Simplify the following expression. Please write your answers in the form of $x + yi$.

(a) $\left(\sin\frac{\pi}{4} + i\cos\frac{\pi}{4}\right)^3$

(b) Find all α such that $\alpha^3 = 1$.

3 State vectors

"Quantum phenomena do not occur in a Hilbert space. They occur in a laboratory." – *Asher Peres.* As we see in lectures, a quantum state for n qubits can be described as a vector in a 2^n dimensional Hilbert space. Mathematically, we use Dirac's bra-ket notation:

$$|\psi\rangle = \sum_{j\in\{0,1\}^n} \alpha_j |j\rangle$$

where $\alpha_j \in \mathbb{C}$ satisfy the normalization condition: $\sum_j |\alpha_j|^2 = 1$.

Task 3.1 (12 pts)

For the following complex vectors, are they valid quantum states? If a vector $|x\rangle$ is a valid quantum state, what is its outer product, $|x\rangle\langle x|$? If not a valid quantum state, what is its inner product, $\langle x|x\rangle$?

(a) $|x\rangle = \frac{1+i}{\sqrt{2}}|0\rangle + \frac{1-i}{\sqrt{2}}|1\rangle$

(b) $|x\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{-i}{\sqrt{2}}|1\rangle$

(c) $|x\rangle = e^{-i\pi/8}|0\rangle + e^{i\pi/8}|1\rangle$

Task 3.2 (10 pts)

A single-qubit quantum state is a vector in the 2-dim Hilbert space. As we discussed in class, a basis for this vector space is a set of vectors such that they span the entire vector space and are linearly independent. An example of such basis is $\{|0\rangle, |1\rangle\}$ (which we call the computational basis). Another example is $\{|+i\rangle, |-i\rangle\}$, where $|+i\rangle = \frac{1}{\sqrt{2}}(|0\rangle + i|1\rangle)$ and $|-i\rangle = \frac{1}{\sqrt{2}}(|0\rangle - i|1\rangle)$. This means that any single-qubit quantum state can be written as a linear combination of the basis vectors. For each of the following quantum states:

- first compute the inner product $\langle\psi|+i\rangle$, and
- then write $\big|\psi\big\rangle$ in the $\{|+i\rangle, |-i\rangle\}$ basis, i.e., find $\alpha, \beta \in \mathbb{C}$ such that $\big|\psi\big\rangle = \alpha|+i\rangle + \beta|-i\rangle$.

(a) $\big|\psi\big\rangle = |0\rangle$

(b) $\big|\psi\big\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$

4 Distinguishing quantum states

In class, we learned that we can distinguish two quantum states via physical experiments, such as applying quantum gates and measurements.

Carol is an experimental physicist. One afternoon, she prepares two qubits (i.e., photon A and photon B) in certain state $\big|\psi_{AB}\big\rangle$ and gives qubit A to Alice and qubit B to Bob. Out of curiosity, Alice and Bob want to figure out if the qubits they receive are the same or different. Since they only received one copy of the qubits each, they cannot simply measure multiple copies of their qubits and reconstruct the distribution of the measurement outcomes. So they turn to you for help.

Task 4.1 (15 pts)

In each of the following scenario, Carol has told you the state of the qubits; your job is to tell Alice and Bob what quantum gates they can perform, so that a computational basis measurement will distinguish the two states *with certainty* (that is, with single shot of measurement). If this is not possible, explain why. (Note: Alice and Bob need to perform the *exact same* set of single-qubit gates on their own qubit, if any. No interaction between the two qubits is allowed.) Allowed quantum gates are:

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}, Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix},$$
$$S = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}, S^\dagger = \begin{bmatrix} 1 & 0 \\ 0 & -i \end{bmatrix}, T = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix}, T^\dagger = \begin{bmatrix} 1 & 0 \\ 0 & e^{-i\pi/4} \end{bmatrix}, \sqrt{X} = \frac{1}{2} \begin{bmatrix} 1+i & 1-i \\ 1-i & 1+i \end{bmatrix}$$

For example, if $|\psi_{AB}\rangle = |+\rangle \otimes |-\rangle$, then it is possible and we can apply H gate then measure in the standard basis, because $H|+\rangle = |0\rangle, H|-\rangle = |1\rangle$.

(a) $|\psi_{AB}\rangle = \left(\frac{|0\rangle+i|1\rangle}{\sqrt{2}}\right) \otimes \left(\frac{|0\rangle-i|1\rangle}{\sqrt{2}}\right)$

(b) $|\psi_{AB}\rangle = \left(\frac{|0\rangle+e^{-i\pi/4}|1\rangle}{\sqrt{2}}\right) \otimes \left(\frac{e^{i\pi/8}|0\rangle+e^{-i\pi/8}|1\rangle}{\sqrt{2}}\right)$

Task 4.2 (⋆ pts)

When qubits are entangled, your job again is to tell Alice and Bob what quantum gates they can perform, so that a computational basis measurement outcome will be different for them. If this is not possible, explain why.

(a) $|\psi_{AB}\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$

(b) $|\psi_{AB}\rangle = \frac{1}{\sqrt{2}}(|+\rangle \otimes |+\rangle + |-\rangle \otimes |-\rangle)$

PROGRAMMING PORTION

This portion of the assignment has a total of 48 *points.*

5 Circuit Simulator

Please read the instructions for how to get started with the programming tasks in Section 1. The goal of these tasks is to understand the basic data structures and operations used in quantum circuits. In the end, we will have a working in-house quantum circuit simulator.

Before attempting the following tasks, you may want to read the starter code in `A1.py` carefully. We have provided a simple implementation of the `Qubit` class, the `Gate` class and the `QuantumCircuit` class. The implementation of the `h` gate and the `measure` operation have already been given.

Task 5.1 (24 pts)

Implement the gate instructions in the `QuantumCircuit` class. In this task, you will extend the `QuantumCircuit` class by implementing the instruction set (i.e., all logical gates supported in the quantum circuit). Each gate method should:

1. Instantiate the `Gate` object using the gate's name (string), the number of qubits it acts on, and its corresponding unitary matrix (as a NumPy array).
   Example:
   ```
   HGate = Gate('h',1,1/np.sqrt(2)*np.array([[1,1],[1,-1]],dtype=complex))
   ```

2. Append the gate to the circuit by calling the `_append()` method of the `QuantumCircuit` class.

For example, in `A1.py`, you will find the Hadamard gate defined as follows:

```
def h(self, qubit):
    # Define Gate by its name, kind (number of qubit), and matrix
    HGate = Gate('h',1,1/np.sqrt(2)*np.array([[1,1],[1,-1]],dtype=complex))
    self._append(HGate, [qubit], [])
    return
```

Notice that the `_append` function takes three arguments: a `Gate` object, a list of qubit indices on which the gate acts, and a list of classical bit indices. For this assignment, you can assume the classical bit list is always empty (`[]`). Usage example: once the Hadamard method is implemented, we can build a simple quantum circuit like this:

```
qc = QuantumCircuit(1, 1)
qc.h(0)
```

Following this pattern, your task is to implement additional gate methods, including the Pauli gates (`X, Y, Z`), Phase gates (`S` and its inverse $\texttt{S}^\dagger$), T gates (`T` and its inverse $\texttt{T}^\dagger$), two-qubit controlled gates (`cx` and `cz`) and three-qubit Toffoli gate (`toffoli`).

Task 5.2 ($\star$ pts)

Implement the `permutation` function that constructs a permutation matrix representing how to reorder qubits. Although this task is optional, the permutation matrix will be very useful for the next task, where you will need to implicitly reorder qubits in multi-qubit operations. The function takes as input a list of `indices` of length n, which describes how to reorder the qubit register `[0,1,2,..., n-1]`, and returns a $2^n \times 2^n$ matrix (as `np.array`). You can assume that `indices` is a valid list containing integers $0,\ldots,n-1$. If `indices[i]==p`, then the qubit at original position i moves to position p. The resulting permutation matrix is a binary matrix with exactly one "1" in each row and one "1" in each column (all other entries are zero).

Concretely,

- `permutation([0,1,2,3])` should return the identity matrix of size $2^4 \times 2^4$.
- `permutation([3,1,0,2])` produces a matrix that, when applied to a basis state $\left|q_0 q_1 q_2 q_3\right\rangle$, reorders the qubit indices to $\left|q_2 q_1 q_3 q_0\right\rangle$.

Hint: think about the meaning of an entry in the permutation matrix: if the entry at column j and row k is 1, that means the j^{th} computational basis state $\left|j\right\rangle$ is mapped to the k^{th} basis state $|k\rangle$. For example, `permutation([2,0,1])` should return a matrix that looks like:

$$\texttt{P} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Task 5.3 (24 pts)

Implement the `tensorizeGate` function. It takes as input a quantum gate (on one, two or three qubits) and a list of qubit indices it acts on. The function returns the corresponding full unitary matrix acting on *all* qubits in `QuantumRegister`) of the circuit. You can assume that the input `gate` object has its matrix defined (dimension $2^k \times 2^k$ where $k = 1,2,3$) and we want to produce a $2^n \times 2^n$ matrix where n is the total number of qubits in the register. We assume the standard ordering of basis states: $|q_0 q_1 q_2 \dots q_{n-1}\rangle$. That means qubit 0 is the leftmost bit in the computational basis representation.

For example, if the `QuantumRegister` has 3 qubits and the quantum gate `cx` acts on qubits `q2` (as control) and `q0` (as target), then `tensorizeGate` should return the $2^3 \times 2^3$ matrix (as an `np.array`) for that operation, with identities on untouched qubits.

```
qc = QuantumCircuit(3, 3)
qc.cx(2, 0)
(op, q_arr, _) = qc.circuit[qc.pc]
tensorized = qc.tensorizeGate(op, q_arr)
```

The above lines of code will result in the following matrix:

$$\texttt{tensorized} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Hint: A neat way to build the full tensorized gate is to use permutation matrices (from the previous task) to shuffle the qubits into the right place:

$$U_{\text{tensorized}} = P^{-1}(G \otimes I_{\text{rest}})P$$

where G is the gate matrix (acting on consecutive qubits), I_{rest} is the identity on the other qubits, and P is a permutation matrix to rearrange the qubits. For example, for `cx([2,0])` in a 3-qubit system, we can write:

$$\texttt{cx(2,0)} = \texttt{permutation([2,0,1])} \cdot (\texttt{cx(0,1)} \otimes I) \cdot \texttt{permutation([1,2,0])}$$

This "sandwiching" trick will simplify your implementation and generalization to any set of control/target qubits.

Task 5.4 ($\star$ pts)

Implement the `simulate` function. This function simulates the quantum circuit from scratch, returning a final quantum state vector (as an `np.array` of `complex` type) if there are no measurements in the circuit, *or* a sampled classical state vector (as an `np.array` of `bool` type) otherwise. Please use the provided `sampleBit` function for random sampling. You may use/modify the `evolveOneStep` function as a helper function for `simulate`. You may assume that all measurements are the last few elements in `circuit` if any. For this assignment, you may also assume that (computational basis) measurement (if present) will be performed to all qubits.