# Programming Quantum Simulation Warm-up

Evan McKinney

August 2020

## Preface

Notes from Chapter 4 of Yanofsky and Mannucci's Quantum Computing for Computer Scientists book. Full python code avaible at https://github.com/evmckinney9/QuantumDrill4

## 1 Basic Quantum Theory

### 1.1 Quantum States

Consider a quantum physical system of a particle confined to a set of discrete positions on a line. Let us suppose that it can only be detected at one of the equally spaced points $\{x_0, x_1, ..., x_{n-1}\}$, where $x_1 = x_0 + \delta x, x_2 = x_1 + \delta x, \ldots$, with $\delta x$ some fixed increment.

We are going to associate the current state of the particle an n-dimensional complex column vector $[c_0, c_1, ..., c_{n-1}]^T$.

The particle being at the point $x_i$ shall be denoted as $|x_i\rangle$, using the Dirac ket notation. To each of these n basic states, we shall associate a column vector:

$$\begin{aligned} |x_0\rangle &\longmapsto [1, 0, ....0]^T \\ |x_1\rangle &\longmapsto [0, 1, ....0]^T \\ &\ldots \\ |x_{n-1}\rangle &\longmapsto [0, 0, ....1]^T \end{aligned} \tag{1}$$

An arbitrary state, which we shall denote as $|\psi\rangle$, will be a linear combination of $|x0\rangle, |x1\rangle, ..., |x_{n-1}\rangle$, by suitable complex weights, $c_0, c_1, ..., c_{n-1}$, known as complex amplitudes.

$$|\psi\rangle = c_0|x_0\rangle + c_1|x_1\rangle + ... + c_{n-1}|x_{n-1}\rangle \tag{2}$$

Thus, every state of our system can be represented by an element of $\mathbb{C}^n$ as

$$|\psi\rangle \longmapsto [c_0, c_1, ..., c_{n-1}]^T \tag{3}$$

We say that the state $|\psi\rangle$ is a superposition of the basic states. $|\psi\rangle$ represents the particle as being simultaneously in all $\{x_0, x_1, ..., x_{n-1}\}$ locations. The complex numbers $c_0, c_1, ..., c_{n-1}$ tell us precisely which superposition our particle is currently in.

The norm square of the complex number $c_i$ divided by the norm squared of $|\psi\rangle$ will tell us the probability that, after observing the particle, we will detect it at the point $x_i$:

$$p(x_i) = \frac{|c_i|^2}{||\psi\rangle|^2} = \frac{|c_i|^2}{\sum_j |c_j|^2} \tag{4}$$

The inner product of the state space gives us a tool to compute complex numbers known as transition amplitudes, which in turn will enable us to determine how likely the state of the system before a specific measurement, will change to another, after measurement has been carried out.

Let

$$|\psi\rangle = \begin{bmatrix} c_0 \\ c_1 \\ \dots \\ c_{n-1} \end{bmatrix} \quad \text{and} \quad |\psi'\rangle = \begin{bmatrix} c'_0 \\ c'_1 \\ \dots \\ c'_{n-1} \end{bmatrix} \tag{5}$$

be two normalized states. We can extract the transition amplitude between state $|\psi\rangle$ and $|\psi'\rangle$. $|\psi\rangle$ will be our state state. The end state will be a row vector whose coordinates will be the complex conjugate of $|\psi'\rangle$ coordinates.
Such a state is called a bra, and will be denoted by $\langle\psi'|$, or equivalently

$$\langle\psi'| = |\psi'\rangle^\dagger = [\overline{c'_0}, \overline{c'_1}, ..., \overline{c'_{n-1}}] \tag{6}$$

To find the transition amplitude we multiply them as matrices:

$$\langle\psi'|\psi\rangle = [\overline{c'_0}, \overline{c'_1}, ..., \overline{c'_{n-1}}] \begin{bmatrix} c_0 \\ c_1 \\ \dots \\ c_{n-1} \end{bmatrix} = \overline{c'_0} \times c_0 + \overline{c'_1} \times c_1 + \ldots + \overline{c'_{n-1}} \times c_{n-1} \tag{7}$$

## 1.2 Summary

- We have learned to associate a vector space to a quantum system. The dimension of this space reflects the amount of basic states of the system.
- States can be superposed, by adding their representing vectors.
- A state is left unchanged if its representing vector is multiplied by a complex scalar.
- The state space has a geometry, given by its inner product. This geometry has a physical meaning: it tells us the likelihood for a given state to transition into another one after being measured. States that are orthogonal to one another are mutually exclusive.

## 1.3 Programming Exercise

Write a program that simulates the quantum system described. The user should be able to specify how many points the particle can occupy. The user will also specify a ket state vector by assigning its amplitudes. The program, when asked the likelihood of finding the particle at a given point, will perform the calculations described. If the user enters two kets, the system will calculate the probability of transitioning from the first ket to the second, after an observation has been made.

```
def simpleQuantumSystem(ket1, ket2=np.array(None)):

    numpoints = len(ket1) #input validation
    if (numpoints > max_points):
        raise Exception("Specified points exceeds maximum")

    if (ket2.shape == ()):
    # calculate the likelihood of finding the particle at a given point

        while (1): #input validation
```

```
            i = int(input(f"Input between 0 and {numpoints-1}: "))
            if i in range(0, numpoints):

                #equation (4) tells us the probability we will detect
                #particle at point x_i, norm square of c_i
                #divided by norm squared of |psi>
                return abs(ket1[i])**2 / (np.linalg.norm(ket1))**2


    else:
        # calculate the probability of transitioning from the first ket
        # to the second, after an observation

        if (ket2.size != ket1.size): #input validation
            raise Exception("Ket2 does not match size of Ket1")

        #equation (7) tells us how to calculate the transition amplitude
        #inner product (using complex conjugate) divided by product of norms
        amplitude = np.vdot(ket2.T, ket1)/(np.linalg.norm(ket1) * np.linalg.norm(ket2))

        #probability is norm squared because eigenvectors
        #constitute an orthognal basis of the state space
        return abs(amplitude)**2
```

## 2 Observables

We can think of a physical system as specified by a double list: on the one hand, its state space, i.e., the collection of all the states it can possibly be found in, and on the other hand, the set of its observables, i.e., the physical quantities that can be observed in each state of the state space.

Each observable may be thought of as a specific question we pose to the system: if the system is current in some given state $|\psi\rangle$, which values can we possibly observe?

To each physical observable there corresponds a hermitian operator. An observable is a linear operator, which means that it maps states to states. If we apply the observable $\Omega$ to the state vector $|\psi\rangle$, the resulting state is now $\Omega|\psi\rangle$. Furthermore, the eigenvectors of $\Omega$ form a basis for the state space.

For the product of two hermitian operators to be hermitian then

$$\langle \Omega_1 \star \Omega_2 \phi, \psi \rangle = \langle \psi, \Omega_1 \star \Omega_2 \psi \rangle. \tag{8}$$

This in turn implies

$$\Omega_1 \star \Omega_2 = \Omega_2 \star \Omega_1, \tag{9}$$

or equivalently, the operator

$$\left[\Omega_1, \Omega_2\right] = \Omega_1 \star \Omega_2 - \Omega_2 \star \Omega_1 \tag{10}$$

must be the zero operator (i.e., the operator that sends every vector to the zero vector).

The operator $\left[\Omega_1, \Omega_2\right]$ is called the commutator of $\Omega_1$ and $\Omega_2$. If the commutator is zero then the product is hermitian.

Recall that hermitian operators behave such that

$$\langle \Omega\phi, \psi \rangle = \langle \phi, \Omega\psi \rangle \tag{11}$$

for each pair $|\psi\rangle, |\phi\rangle$. This immediately derives that $\langle \Omega\phi, \psi\rangle$ is a real number for each $|\psi\rangle$, which we shall denote as $\langle\Omega\rangle_\psi$ (the subscript points to the fact that this quantity depends on the state vector). $\langle\Omega\rangle_\psi$ is the expected value of observing $\Omega$ repeatedly on the same state $\psi$.

Suppose that $\lambda_0, \lambda_1, \ldots, \lambda_{n-1}$ is the list of eigenvalues of $\Omega$. Let us prepare our quantum system so that it is in state $|\psi\rangle$ and let us observe the value of $\Omega$. We are going to obtain one of the eigenvalues. Repeat n times, and keep track of what was observed each time. At the end of the experiment, the eigenvalue $\lambda_i$ has been seen $p_i$ times, where $0 \leq p_i \leq n$.

If n is sufficiently large, then the estimated expected value of $\Omega$ will be approach $\langle \Omega\phi, \psi\rangle$ calculated by

$$\lambda_0 \times \frac{p_0}{n} + \lambda_1 \times \frac{p_1}{n} + \ldots + \lambda_{n-1} \times \frac{p_{n-1}}{n} \tag{12}$$

We may be interested in knowing the spread of the distribution around its expected value, i.e., the variance of the distribution on the set of eigenvalues. First introduce the hermitian operator

$$\Delta_\psi(\Omega) = \Omega - \langle\Omega\rangle_\psi I \tag{13}$$

(I is the identity operator). The operator $\Delta_\psi(\Omega)$ acts on a generic vector $|\phi\rangle$ in the following fashion:

$$\Delta_\psi(\Omega)|\phi\rangle = \Omega(|\phi\rangle) - (\langle\Omega\rangle_\psi)|\phi\rangle \tag{14}$$

So $\Delta_\psi(\Omega)$ is the demeaned version of $\Omega$.

We can now define the variance of $\Omega$ at $|\psi\rangle$ as the expectation value of $\Delta_\psi(\Omega)$ squared:

$$Var_\psi(\Omega) = \langle(\Delta_\psi(\Omega)) \star (\Delta_\psi(\Omega))\rangle_\psi \tag{15}$$

remembering that

$$Var(X) = E((X-\mu)^2) = E((X-\mu)(X-\mu)) \tag{16}$$

where E is the expected value function.

**Heisenberg's Uncertainty Principle**

Let us begin with two observables, represented by the two hermitians $\Omega_1$ and $\Omega_2$, and a given state, say, $|\psi\rangle$. We can compute the variance of $\Omega_1$ and $\Omega_2$ on $|\psi\rangle$, obtaining $Var_\psi(\Omega_1)$ and $Var_\psi(\Omega_2)$. We have two observables, and our goal is to simultaneously minimize their variances, thereby getting a sharp outcome for both. If there were no correlation in the variances, we could expect a very sharp measure of both observables on some state. This is not the case.

The product of the variances of two arbitrary hermitian operators on a given state is always greater than or equal to one-fourth the square of the expected value of their commutator:

$$Var_\psi(\Omega_1) \times Var_\psi(\Omega_2) \geq \frac{1}{4}|\langle\big[\Omega_1, \Omega_2\big]\rangle_\psi|^2 \tag{17}$$

The commutator measures how good a simultaneous measure of two observables can possibly be. If the commutator happens to be zero, there is no limit to accuracy. In quantum mechanics, plenty of operators do not commute, e.g., directional spin operators.

Consider the quantum system given by the pair position-momentum. The image of $|\psi\rangle$ in the position basis is very different from the one associated with the momentum eigenbasis. The position eigenbasis is made of Dirac's deltas. Therefore, $|\psi\rangle$ is decomposed into a weighted sum of peaks. The momentum eigenbasis, on the other hand, is made of sinusoids, whose position is totally undetermined. The commutator of the position-momentum pair captures this and is thus not zero. IF we can pin down the particle position at a given point in time (i.e., if the variance of its position operator is very small), we are at a loss as to its momentum (i.e., the variance of its momentum operator is very big), and vice versa.

## 2.1 Summary

- Observables are represented by hermitian operators. The result of an observation is always an eigenvalue of the hermitian.
- The expression $\langle\psi|\Omega|\psi\rangle$ represents the expected value of observing $\Omega$ on $|\psi\rangle$.
- Observables in general do not commute. This means that the order of observation matters. Moreover, if the commutator of two observables is not zero, there is an intrinsic limit to our capability of measuring their values simultaneously.

## 2.2 Programming Exercise

Continue your simulation of a quantum system by adding observables to the picture: the user will input a square matrix of appropriate size, and a ket vector. The program will verify that the matrix is hermitian, and if so, it will calculate the mean value and the variance of the observables on the given state.

```
def verifyHermitian(matrix):
    if not square(matrix): #input validation
        raise Exception("Not a square matrix")

    X = np.matrix(matrix)
    #iff conjugate transpose equals itself
    return (X.getH() == X).all()

def observableMeanValue(observable, ket):
    #see example 4.2.5 from book

    #bra is row vector complex conjugate
    bra = np.matmul(observable, ket).T

    #vdot will take the complex conjugate of bra
    return np.vdot(bra, ket)


def observableVariance(observable, ket):
    mean = observableMeanValue(observable,ket)
    #hermitian operator from equation (13)
    h_o = observable - mean * np.identity(len(observable))
    h_o_squared = np.matmul(h_o, h_o)

    #calculate variance using equation (15)
    #see example 4.2.6 from book
    return np.matmul(np.matmul(np.conj(ket.T), h_o_squared), ket)
```

# 3 Measuring

The act of carrying out an observation on a given physical system is called measuring. Measuring is the process consisting of asking a specific question and receiving a definite answer.

Systems do get perturbed and modified as a result of measuring them. Furthermore, only the probability of observing specific values can be calculated: measurement is inherently a non-deterministic process.

An observable can only assume one of its eigenvalues as the result of an observation. Let $\Omega$ be an observable and $|\psi\rangle$ be a state. If the result of measuring $\Omega$ is the eigenvalue $\lambda$, the state after measurement will always be an eigenvector corresponding to $\lambda$.

The probability that a normalized start state $|\psi\rangle$ will transition to a specific eigenvector $|e\rangle$ is given by the square of the inner product of the two states: $|\langle e|\psi\rangle|^2$. This expression is the projection of $|\psi\rangle$ along $|e\rangle$.

Let us recall that the normalized eigenvectors of $\Omega$ constitute an orthogonal basis of the state space. Therefore, we can express $|\psi\rangle$ as a linear combination in this basis:

$$|\psi\rangle = c_0|e_0\rangle + c_1|e_1\rangle + \dots c_{n-1}|e_{n-1}\rangle \tag{18}$$

Now, let us compute the mean:

$$\langle\Omega\rangle_\psi = \langle\Omega\psi, \psi\rangle = |c_0|^2\lambda_0 + |c_1|^2\lambda_1 + \dots + |c_{n-1}|^2\lambda_{n-1} \tag{19}$$

As we can now see, $\langle\Omega\rangle_v$ is precisely the mean value of the probability distribution

$$(\lambda_0, p_0), (\lambda_1, p_1), , \dots, (\lambda_{n-1}, p_{n-1}) \tag{20}$$

where each $p_i$ is the square of the amplitude of the collapse into the corresponding eigenvector.

## 3.1 Summary

- The end state of the measurement of an observable is always one of its eigenvectors.
- The probability for an initial state to collapse into an eigenvector of the observable is given by the length squared of the project.
- When we measure several observables, the order of measurements matters

## 3.2 Programming Exercise

Next step in the simulation: when the user enters an observable and a state vector, the program will return the list of eigenvalues of the observable, and the mean value of the observable on the state, and the probability that the state will transition to each one of the eigenstates. Optional: plot the corresponding probability distribution.

```
def observableEigenVectors(observable):
    _, eigenvectors = np.linalg.eigh(observable)
    return eigenvectors

def observableEigenValues(observable):
    eigenvalues, _ = np.linalg.eigh(observable)
    return eigenvalues

def eigenstatesProability(observable, ket):
    #return the probability that the state will transition to each of the eigenstates
    eigenstates = observableEigenVectors(observable)
    eigenvalues = observableEigenValues(observable)
    pairs = []
    for eigenvalue, eigenstate in zip(eigenvalues, eigenstates.T):
        #eigenstate should be a column vector
```

```
        eigenstate.shape = (eigenstate.size,1)
        probability = simpleQuantumSystem(ket, eigenstate)
        pairs.append((eigenvalue, probability))
    return pairs

def eigenvalueMeanValue(observable, ket):
    #calculates expected value of observable on a state
    #using eigenvalues and probabilities
    pairs = eigenstatesProability(observable, ket)
    mean = 0
    for i in pairs:
        mean += i[0] * i[1]
    return mean
```

# 4 Dynamics

In reality, quantum systems do evolve over time, and thus we need to consider quantum dynamics. Just as hermitian operators represent physical observables, unitary operators introduce dynamics in the quantum arena.

The evolution of a quantum system (that is not a measurement) is given by a unitary operator or transformation. That is, if U is a unitary matrix that represents a unitary operator and $|\psi(t)\rangle$ represents a state of the system at time t, then

$$|\psi(t+1)\rangle = U|\psi(t)\rangle \tag{21}$$

will represent the system at time t+1.

Unitary transformations are closed under composition and inverse. One says that the set of unitary transformations constitutes a group of transformations with respect to composition.

We are now going to see how dynamics is determined by unitary transformations: assume we have a rule, $\mathfrak{U}$, that associates with each instant of time $t_0, t_1, \ldots, t_{n-1}$ a unitary matrix $\mathfrak{U}[t_0], \mathfrak{U}[t_1], \ldots, \mathfrak{U}[t_{n-1}]$.

Let us start with an initial state vector $|\psi\rangle$. We can apply $\mathfrak{U}[t_0]$ to $|\psi\rangle$, then apply to the result, and so forth. We will obtain a sequence of state vectors

$$\begin{aligned} &\mathfrak{U}[t_0]|\psi\rangle, \\ &\mathfrak{U}[t_1]\mathfrak{U}[t_0]|\psi\rangle, \\ &\ldots, \\ &\mathfrak{U}[t_{n-1}]\mathfrak{U}[t_{n-2}]\ldots\mathfrak{U}[t_0]|\psi\rangle. \end{aligned} \tag{22}$$

Such a sequence is called the orbit of $|\psi\rangle$ under the action of $\mathfrak{U}[t_i]$ at the time clicks $t_0, t_1, \ldots, t_{n-1}$.

By applying the inverses of $\mathfrak{U}[t_0], \mathfrak{U}[t_1], \ldots, \mathfrak{U}[t_{n-1}]$ in reverse or the the quantum system will evolve backwards as it is symmetric with respect to time.

A quantum computer shall be placed into an initial state $|\psi\rangle$, and we shall then apply a sequence of unitary operators to the state. When we are done, we will measure the output and get a final state.

### Schrödinger equation

How is the sequence $\mathfrak{U}[t_i]$ of unitary transformations actually selected in real-life quantum mechanics?

$$\frac{|\psi(t+\delta t)\rangle - |\psi(t)\rangle}{\delta t} = -i\frac{2\pi}{\hbar}\mathcal{H}|\psi(t)\rangle. \tag{23}$$

Energy is an observable, and therefore for a concrete quantum system is it possible to write down a hermitian matrix representing it. This observable is called the hamiltonian of the system, indicacted by $\mathcal{H}$.

The Schrödinger equation states that the rate of variation of the state vector $|\psi(t)\rangle$ with respect to time at the instant t is equal (up to the scalar factor $\frac{2\pi}{\hbar}$) to $|\psi(t)\rangle$ multiplied by the operator $-i * \mathcal{H}$. By solving the equation with some initial conditions one is able to determine the evolution of the system over time.

### 4.1 Summary

- Quantum dynamics is given by unitary transformations
- Unitary transformations are invertible; thus, all closed system dynamics are reversible in time (as long as no measurement is involved).
- The concrete dynamics is given by the Schrodinger equation. which determines the evolution of a quantum system whenever its hamiltonian is specified

### 4.2 Programming Exercise

Add dynamics to your computer simulation of the particle on a grid: the user should input a number of time steps n, and a corresponding sequence of unitary matrices $U_n$ of the appropriate size. The program will then compute the state vector after the entire sequence $U_n$ has been applied.

```
def dynamicSimulation(ket, matrix_sequence, timesteps=-1,):
    if timesteps == -1: #input validation
        timesteps = len(matrix_sequence)
    if timesteps > len(matrix_sequence) or timesteps < 0:
        raise Exception("invalid timesteps")

    if timesteps == 0:
        return ket

    unitary = matrix_sequence[1]
    for i in range(1,timesteps):
        unitary = np.matmul(unitary, matrix_sequence[i])
    return np.matmul(unitary,ket)
```

## 5 Assembling Quantum Systems

Recall the simple quantum system of a particle moving in a confined one-dimensional grid on the set of points $\{x_0, x_1, \ldots, x_{n-1}\}$. Now, let us suppose that we are dealing with two particles confined to the grid. The points on the grid that can be occupied by the first particle will be $\{x_0, x_1, \ldots, x_{n-1}\}$ and the second particle can be at the points $\{y_0, y_1, \ldots, y_{n-1}\}$.

Assembling quantum systems means tensoring the state space of their constituents. Assume we have two independent quantum systems Q and Q', represented respectively by the vector spaces $\mathbb{V}$ and $\mathbb{V}'$. The quantum system obtained by merging Q and Q' will have tensor product $\mathbb{V} \otimes \mathbb{V}'$ as a state space.

We can assemble as many systems as we like. The tensor product of vector spaces is associative, so we can progressively build larger and larger systems:

$$\mathbb{V}_0 \otimes \mathbb{V}_1 \otimes \ldots \otimes \mathbb{V}_k \tag{24}$$

There are $n \times m$ possible basic states:

$$|x_i\rangle \otimes |y_j\rangle, \text{meaning the first particle is at } x_i \text{and the second particle is at } y_j.$$

Now, let us write the generic state vector as a superposition of the basic states:

$$|\psi\rangle = c_{0,0}|x_0\rangle \otimes |y_0\rangle + \ldots + c_{i,j}|x_i\rangle \otimes |y_j\rangle + \ldots + c_{n-1,m-1}|x_{n-1}\rangle \otimes |y_{m-1}\rangle \tag{25}$$

which is a vector in the $(n \times m)$-dimensional complex space $\mathbb{C}^{n\times m}$.

The quantum amplitude $|c_{i,j}|$ squared will give us the probability of finding the two particles at positions $x_i$ and $y_j$, respectively.

**Entanglement**

Assembled complex systems cannot be understood completely in terms of their constituents. Consider the state

$$|\psi\rangle = |x_0\rangle \otimes |y_0\rangle + |x_1\rangle \otimes |y_1\rangle$$

equivalently,

$$|\psi\rangle = 1|x_0\rangle \otimes |y_0\rangle + 0|x_0\rangle \otimes |y_1\rangle + 0|x_1\rangle \otimes |y_0\rangle + 1|x_1\rangle \otimes |y_1\rangle. \tag{26}$$

Then the vector representing the first particle on the line can be written as $c_0|x_0\rangle + c_1|x_1\rangle$ and the vector representing the second particle on the line can be written as $c_0'|y_0\rangle + c_1'|y_1\rangle$.

Therefore, if $|\psi\rangle$ came from the tensor product of the two subsystems, we would have

$$\begin{aligned}(c_0|x_0\rangle + c_1|x_1\rangle) \otimes (c_0'|y_0\rangle + c_1'|y_1\rangle) =& c_0c_0'|x_0\rangle \otimes |y_0\rangle + c_0c_1'|x_0\rangle \otimes |y_1\rangle \\ &+ c_1c_0'|x_1\rangle \otimes |y_0\rangle + c_1c_1'|x_1\rangle \otimes |y_1\rangle\end{aligned} \tag{27}$$

From Equation (26) this implies that $c_0c_0' = c_1c_1' = 1$ and $c_0c_1' = c_1c_0' = 0$ which has no solution. Conclude that $|\psi\rangle$ cannot be rewritten as a tensor product.

The individual states of the two particles are entangled. The first particle has a 50-50 chance of being found at the position $x_0$ or at $x_1$. If it is found at position $x_0$, because the term $|x_0\rangle \otimes |y_1\rangle$ has a 0 coefficient, we conclude the second particle can only be found in position $y_0$. Similarly, if the first particle is found in position $x_1$, the second particle must be in position $y_1$.

States that can be broken into the tensor product of states from the constituent subsystems are called separable states, whereas states that are unbreakable are referred to as entangled states. Entanglement plays a central role in algorithm design, cryptography, teleportation, and decoherence.

## 5.1 Summary

- We can use the tensor product to build complex quantum systems out of simpler ones.
- The new system cannot be analyzed simply in terms of states belonging to its subsystems. An entire set of new states has been created, which cannot be resolved into their constituents.

## 5.2 Programming Exercise

Expand the simulation of the last sections by letting the user choose the number of particles.

```
def multiQuantumSystem(*multi_stateSpace):
    #returns a tensor product of the vector spaces
    num_particles = len(multi_stateSpace)
    if num_particles <= 0 or num_particles > max_particles:
        raise Exception("invalid number of particles")

    tensor_product = multi_stateSpace[0]

    for i in range(1, num_particles):
        tensor_product = np.kron(tensor_product, multi_stateSpace[i])

    return tensor_product
```