Welcome to Quantum

James Daniel Whitfield

Department of Physics and Astronomy, Dartmouth College

December 14, 2020

Outline

1. Probability
    - States
    - Kinematics: change of basis
    - Kinematics: transforming states
2. Quantum
    - States
    - Kinematics: change of basis
    - Kinematics: transforming states
3. Application: Quantum computing

Outline

1. Probability
   - States
   - Kinematics: change of basis
   - Kinematics: transforming states
2. Quantum
   - States
   - Kinematics: change of basis
   - Kinematics: transforming states
3. Application: Quantum computing

Probability

States

Probability state, $\vec{p}$

In general, $\vec{p}$ is a valid state if:

- $\|\vec{p}\|_1 = \sum_i |p_i| = 1$
- $p_k \in \mathcal{R}$
- $p_k \geq 0$

p_i is the probability of the ith outcome resulting from a measurement.

In [1]:

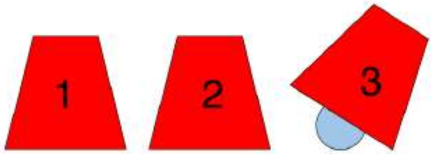
```
import numpy as np

#orthogonal vector corresponding to possible outcomes
e0 = np.matrix([[1],[0],[0]])
e1 = np.matrix([[0],[1],[0]])
e2 = np.matrix([[0],[0],[1]])

#probabilities of possible outcomes
p0 = 0.05
```

```
p1 = 0.15
p2 = 0.8

#probability vector
p = p0 * e0 + p1 * e1 + p2 * e2
```

Probability

Change of basis (kinematics)

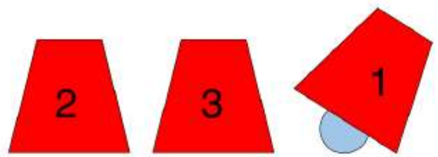
In probability, we can change the basis of the sample space by *permuting* passively i.e. without doing anything



Probability

Change of basis (kinematics)

In probability, we can change the basis of the sample space by *permuting* passively i.e. without doing anything



$$\hat{e}_k \xmapsto{\sigma \in S_N} \hat{e}_{\sigma(k)}$$

$$P^\sigma \cdot \hat{e}_k = \hat{e}_{\sigma(k)}$$

Probability

Change of basis (kinematics)

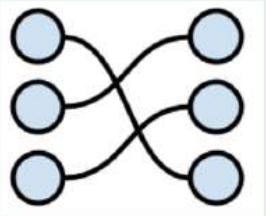
In probability, we can change the basis of the sample space by *permuting*

$$\hat{e}_k \xmapsto{\sigma \in S_N} \hat{e}_{\sigma(k)}$$

$$P^\sigma \cdot \hat{e}_k = \hat{e}_{\sigma(k)}$$

Example: $\sigma = (021) \in S_3$

$$\begin{aligned} \sigma(0) &= 2 \\ \sigma(1) &= 0 \\ \sigma(2) &= 1 \end{aligned}$$



In [4]:

```
sig=[2,0,1]


#A matrix representation of the permutation
P_021=np.matrix(np.zeros((3,3)))
P_021[sig[0],0]=1
P_021[sig[1],1]=1
P_021[sig[2],2]=1
```

```
print(P_021)
```

```
[[0. 1. 0.]
 [0. 0. 1.]
 [1. 0. 0.]]
```

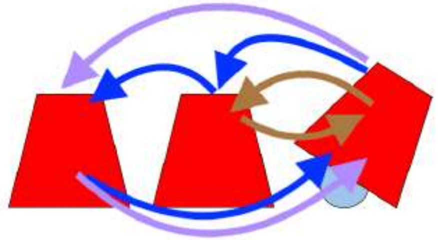
In [6]:

```
P_021@p
```

Out[6]:

```
matrix([[0.15],
        [0.8 ],
        [0.05]])
```

If instead of merely changing the basis, we do so stochastically i.e. with some probability w_σ for each permutation σ.



$$\vec{p}(f) = \left(\sum_\sigma w_\sigma P^\sigma \right) \vec{p}(i)$$

Probability

Transforming states (kinematics)

If instead of merely changing the basis, we do so stochastically i.e. with some probability w_σ for each permutation σ.

$$\vec{p}(f) = \left(\sum_\sigma w_\sigma P^\sigma \right) \vec{p}(i)$$

For example, applying $P^{(132)}$ with probability $w_{(132)} = \frac{1}{2}$ or doing nothing with $w_{id} = \frac{1}{2}$ would give

$$\left(\frac{1}{2} P^{(021)} + \frac{1}{2} \mathbb{1} \right) \vec{p}(i) = \frac{1}{2} \begin{bmatrix} p_1 \\ p_2 \\ p_0 \end{bmatrix} + \frac{1}{2} \begin{bmatrix} p_0 \\ p_1 \\ p_2 \end{bmatrix} = \begin{bmatrix} 0.100 \\ 0.475 \\ 0.425 \end{bmatrix}$$

In [8]:

```
w1=.5
w2=1-w1

(w1* P_021 + w2 * np.eye(3)) @ p
```

Out[8]:

```
matrix([[0.1  ],
        [0.475],
        [0.425]])
```

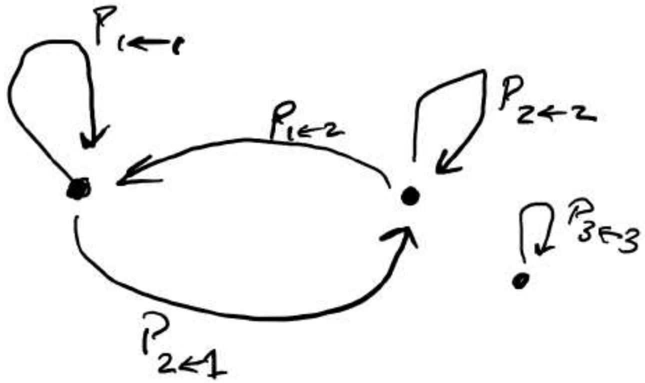
Probability

Transforming states (kinematics)

$P_{j\leftarrow i}$ is probability to transfer from i to j in unit time

- Condition $\sum_j P_{j\leftarrow i} = 1$ follows from definition.
- Natural composition rule follows from definition

$$p_j(f) = \sum_k P_{j\leftarrow k} p_k(i)$$



Probability

Transforming states (kinematics)

$P_{j \leftarrow i}$ is probability to transfer from i to j in unit time

- Condition $\sum_j P_{j \leftarrow i} = 1$ follows from definition.
- Natural composition rule follows from definition

$$p_j(f) = \sum_k P_{j \leftarrow k} p_k(i)$$

$$\vec{p}(f) = \begin{bmatrix} \sum_k P_{0 \leftarrow k} p_k(i) \\ \sum_k P_{1 \leftarrow k} p_k(i) \\ \vdots \\ \sum_k P_{n \leftarrow k} p_k(i) \end{bmatrix} = \begin{bmatrix} P_{0 \leftarrow 0} & P_{0 \leftarrow 1} & \dots & P_{0 \leftarrow n-1} \\ P_{1 \leftarrow 0} & P_{1 \leftarrow 1} & \dots & P_{1 \leftarrow n-1} \\ \vdots & & & \vdots \\ P_{n \leftarrow 0} & P_{n \leftarrow 1} & \dots & P_{n-1 \leftarrow n-1} \end{bmatrix} \begin{bmatrix} p_0(i) \\ p_1(i) \\ \vdots \\ p_{n-1}(i) \end{bmatrix}$$

This arrangement yields the natural definition of matrix multiplication.

Matrix multiplication

Transforming states (kinematics)

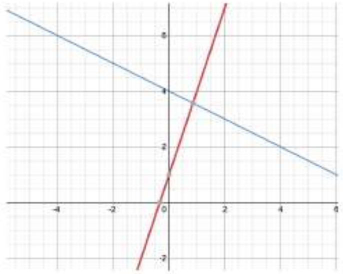
$$\vec{p}(f) = \begin{bmatrix} \sum_k P_{0\leftarrow k} p_k(i) \\ \sum_k P_{1\leftarrow k} p_k(i) \\ \vdots \\ \sum_k P_{n\leftarrow k} p_k(i) \end{bmatrix} = \begin{bmatrix} P_{0\leftarrow 0} & P_{0\leftarrow 1} & \dots & P_{0\leftarrow n-1} \\ P_{1\leftarrow 0} & P_{1\leftarrow 1} & \dots & P_{1\leftarrow n-1} \\ \vdots & & & \vdots \\ P_{n\leftarrow 0} & P_{n\leftarrow 1} & \dots & P_{n-1\leftarrow n-1} \end{bmatrix} \begin{bmatrix} p_0(i) \\ p_1(i) \\ \vdots \\ p_{n-1}(i) \end{bmatrix}$$

Matrix multiplication for matrix M and vector $\vec{x}$

$$\vec{y}_i = (M\vec{x})_i = \sum_{ij} M_{ij} \vec{x}_j$$

Matrix multiplication

Transforming states (kinematics)



Matrix multiplication for matrix M and vector $\vec{x}$

$$\vec{y}_i = (M\vec{x})_i = \sum_{ij} M_{ij}\vec{x}_j$$

Linear algebra: The rich subject of lines

Consider two lines (i.e. no x^2 no y^3): $Ax + B = y$ and $Cx + D = y$

$$\begin{aligned} Ax + (-1)y &= -B \\ Cx + (-1)y &= -D \end{aligned} \tag{1}$$

$$\begin{bmatrix} A & -1 \\ C & -1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} -B \\ -D \end{bmatrix} \tag{2}$$

Questions on probability?

Up next
Quantum Lift

I now invite the reader to reflect on the following aspects of probability theory before embarking on the extension to quantum.

- Meaning of measurement and the subsequent updating of probabilities
- Interpretation of the probability function
    - The probability function as knowledge (epistemic)
    - The probability function as an actual thing (ontic)

Outline

Quantum lift of probability vectors

States

Quantum lift

$$\vec{p} = \sum p_k \hat{e}_k \mapsto \Lambda_p = \mathtt{diag}(\vec{p}) = \sum_k p_k \hat{e}_k \hat{e}_k^\dagger$$

In [11]:

```
# need to flatten the vector before using diag
# if you have numpy array (instead of a vector)
# then np.diag will work directly

dm_p = np.diagflat(p)

print(dm_p)
```

```
[[0.05 0.   0.  ]
 [0.   0.15 0.  ]
 [0.   0.   0.8 ]]
```

Quantum lift of probability vectors

States

Quantum lift

$$\vec{p} = \sum p_k \hat{e}_k \mapsto \Lambda_p = \mathtt{diag}(\vec{p}) = \sum_k p_k \hat{e}_k \hat{e}_k^\dagger$$

Here, we have a few organizational rules and index keeping: In finite dimensional settings, the $\dagger$ is the conjugate transpose.[1]

$$\hat{e}_1^\dagger = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}^\dagger = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}$$

[1] The conjugate of imaginary number $a + bi$ is $(a + bi)^* = (a - bi)$ for complex numbers

In [16]:

```
# np matricies have attribute A.H for conjugate transpose
# alternatively one can use A.transpose().conjugate

#print(e1.conjugate().transpose())
print(e1.H)
```

```
[[0 1 0]]
```

Quantum lift of probability vectors

States

Quantum lift

$$\vec{p} = \sum p_k \hat{e}_k \mapsto \Lambda_p = \mathtt{diag}(\vec{p}) = \sum_k p_k \hat{e}_k \hat{e}_k^\dagger$$

Here, we have a few organizational rules and index keeping: In finite dimensional settings, the $\dagger$ is the conjugate transpose.[1]

$$\text{So } e_1^\dagger e_1 = 1,\ e_0^\dagger e_2 = 0 \text{ and } e_2 e_0^\dagger = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

$$\text{and } (e_2 e_0^\dagger)^\dagger = e_0 e_2^\dagger = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

In [25]:

```
# use the @ for matrix multiplication in numpy

print("e1^dag e1 =",e1.H @ e1)
print(" ")
print("e0^dag e2 =", e0.H @ e2)
print(" ")
print("e2 e0^dag =")
print(e2 @ e0.H)
```

```python
print(" ")
print("e0 e2^dag =")
print(e0 @ e2.H)
```

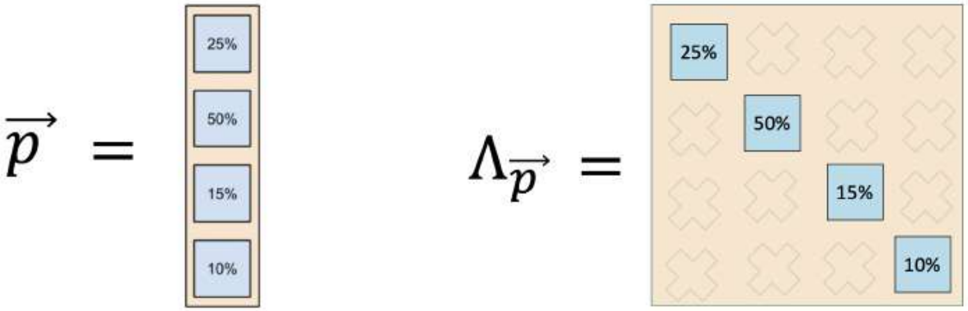
```
e1^dag e1 = [[1]]

e0^dag e2 = [[0]]

e2 e0^dag =
[[0 0 0]
 [0 0 0]
 [1 0 0]]

e0 e2^dag =
[[0 0 1]
 [0 0 0]
 [0 0 0]]
```

Quantum lift of probability vectors

States

Quantum lift

$$\vec{p} = \sum p_k \hat{e}_k \mapsto \Lambda_p = \mathrm{diag}(\vec{p}) = \sum_k p_k \hat{e}_k \hat{e}_k^\dagger$$



In [45]:

```
p_vec = [.25, .5, .15, .1]
L_p = np.diag(p_vec)

print(p_vec)
print(" ")
print(L_p)
```

```
[0.25, 0.5, 0.15, 0.1]
```

```
[[0.25 0.   0.   0.  ]
 [0.   0.5  0.   0.  ]
 [0.   0.   0.15 0.  ]
 [0.   0.   0.   0.1 ]]
```

Quantum

States

Probability state, $\vec{p}$

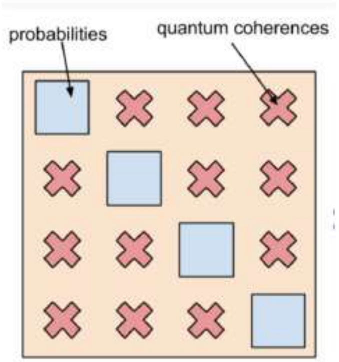
In general, $\vec{p}$ is a valid state if:

- $\|\vec{p}\|_1 = 1$
- $p_k \in \mathcal{R}$
- $p_k \geq 0$

Quantum state, ρ

In general, ρ is a valid state if:

- $\|\rho\|_1 = Tr|\rho| = 1$
- $\rho = \rho^\dagger$
- $\vec{v}^\dagger \cdot \rho \cdot \vec{v} \geq 0$ for all $\vec{v}$



For our quantum lift, $Tr(\Lambda_p) = 1$ and $(\Lambda_p)_{kk} \geq 0$ and $\Lambda_p = \Lambda_p^\dagger$

In [59]:

```
# the unitary Fourier transform matrix
from scipy.linalg import dft
U=np.matrix(dft(3)/np.sqrt(3))
np.set_printoptions(suppress=True,precision=4)
```

```python
#here dm stands for density matrix
dm = U @ dm_p @ U.H

#check that dm is a valid quantum state
print(dm)
print(" ")
print(np.trace(dm).real)
print(" ")
print(np.linalg.eig(dm)[0].real)
print(" ")
print(dm - dm.H)
```

```
[[ 0.3333+0.j      -0.1417-0.1876j -0.1417+0.1876j]
 [-0.1417+0.1876j  0.3333+0.j      -0.1417-0.1876j]
 [-0.1417-0.1876j -0.1417+0.1876j  0.3333+0.j    ]]

1.0

[0.8  0.05 0.15]

[[ 0.+0.j  0.+0.j  0.+0.j]
 [ 0.+0.j  0.+0.j  0.+0.j]
 [ 0.+0.j -0.+0.j  0.+0.j]]
```

Quantum

Measurement of states

Decoherent projection of quantum state

$$\mathcal{E}_{\hat{e}}(\rho) = \sum_k (\hat{e}_k \hat{e}_k^\dagger)\rho(\hat{e}_k \hat{e}_k^\dagger) = \sum \mathrm{P}_k \; \rho \; \mathrm{P}_k$$

Both $P_k^2 = P_k = (\hat{e}_k \hat{e}_k^\dagger)$ and $\mathcal{E}_{\hat{e}}^2 = \mathcal{E}_{\hat{e}}$ i.e. are projective.

With respect to the $\hat{e}$ *basis*, there are no more coherences (i.e. $\mathcal{E}_{\hat{e}}(\rho) = \Lambda_{\vec{p}}$ for some valid $\vec{p}$). Then we treat $\vec{p}$ with "ordinary" probability.

In [60]:

```
print(dm)
print(" ")
print(   (e0 @ e0.H) @ dm @ (e0 @ e0.H)
       + (e1 @ e1.H) @ dm @ (e1 @ e1.H)
       + (e2 @ e2.H) @ dm @ (e2 @ e2.H))
```

```
[[ 0.3333+0.j      -0.1417-0.1876j -0.1417+0.1876j]
 [-0.1417+0.1876j  0.3333+0.j      -0.1417-0.1876j]
 [-0.1417-0.1876j -0.1417+0.1876j  0.3333+0.j    ]]
```

```
[[0.3333+0.j 0.     +0.j 0.     +0.j]
 [0.     +0.j 0.3333+0.j 0.     +0.j]
 [0.     +0.j 0.     +0.j 0.3333+0.j]]
```

Quantum

Change of basis (kinematics)

In probability, we permuted the sample space to change the basis:

Probability	Quantum
$\vec{p}(f) = P^{\sigma}\vec{p}(i)$	$P^{\sigma} \cdot \left(\Lambda_{p(i)}\right) \cdot P^{\sigma\dagger} = \Lambda_{p(f)}$

In [65]:

```
print(P_021 @ dm_p @ P_021.H)
print(" ")
print(P_021 @ p)
```

```
[[0.15 0.   0.  ]
 [0.   0.8  0.  ]
 [0.   0.   0.05]]

[[0.15]
 [0.8 ]
 [0.05]]
```

Quantum

Change of basis (kinematics)

In probability, we permuted the sample space to change the basis:

Probability	Quantum
$\vec{p}(f) = P^\sigma \vec{p}(i)$	$P^\sigma \cdot (\Lambda_{p(i)}) \cdot P^{\sigma\dagger} = \Lambda_{p(f)}$
	$\Lambda_{p(f)} = U \cdot (\Lambda_{p(i)}) \cdot U^\dagger$

The quantum state space as a matrix space allows for continuous changes of basis via unitary matrices.[1]

- $U^\dagger U = 1$
- $\|U\vec{x}\|_2 = \|\vec{x}\|_2 = \sqrt{\sum_i |x_i|^2}$ for all $\vec{x}$

Exercise

Permutations are unitary

[1] You may again think of this as either as an active transformation that "happened to the system" or a passive "relabelling" of a fixed state.

In [64]:

```python
# the unitary Fourier transform matrix
from scipy.linalg import dft
```

```python
U=np.matrix(dft(3)/np.sqrt(3))
np.set_printoptions(suppress=True,precision=4)

# An examples from earlier
print(U @ U.H)
print(" ")
print(P_021 @ P_021.H)
```

```
[[ 1.+0.j -0.+0.j  0.+0.j]
 [-0.-0.j  1.-0.j -0.+0.j]
 [ 0.-0.j -0.-0.j  1.-0.j]]

[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

Quantum

Transforming states (kinematics)

Deterministic change of basis

$$\mathcal{E}_U(\rho) = U\rho U^\dagger$$

Stochastic change of basis

Probability	Quantum
$\vec{p}(f) = \sum_\sigma w_\sigma (P^\sigma \vec{p}(i))$	$\sum w_\sigma \left(P^\sigma \cdot \Lambda_{p(i)} \cdot P^{\sigma\dagger}\right) = \Lambda_{M\vec{p}(i)}$

$$\Lambda_{Mp(i)} = \sum w_\alpha U_\alpha \Lambda_{p(i)} U_\alpha^\dagger$$

Stochastic change of basis

$$\mathcal{E}_{\{w_\alpha, U_\alpha\}}(\rho) = \sum w_\alpha U_\alpha \rho U_\alpha^\dagger$$

In [73]:

```
w1=.5
w2=1-w1

dm_pf =  w1 * P_021 @ dm_p @ P_021.H\
       + w2 * np.eye(3) @ dm_p @ np.eye(3).T
pf    = (w1* P_021 @ p + w2 * np.eye(3) @ p )

print(dm_pf)
```

```
print(" ")
print(pf)
```

```
[[0.1   0.    0.   ]
 [0.    0.475 0.   ]
 [0.    0.    0.425]]

[[0.1  ]
 [0.475]
 [0.425]]
```

Quantum

Transforming states (kinematics)

Type 1: Projective measurements

$$\mathcal{E}_{\hat{e}}(\rho) = \sum_j (\hat{e}_j \hat{e}_j^\dagger)\rho(\hat{e}_j \hat{e}_j^\dagger)$$

Type 2: Stochastic change of basis

$$\mathcal{E}_{\{w_\alpha, U_\alpha\}}(\rho) = \sum w_\alpha \; U_\alpha \rho U_\alpha^\dagger$$

General case: Kraus (operator sum) representation

$$\mathcal{E}(\rho) = \sum E_k \rho E_k^\dagger \text{ with } \sum E_k^\dagger E_k = 1$$

Standard elementary formalism

Usual treatment begins with wave functions: $\vec{\psi}$ where

$$U_t\vec{\psi} = \vec{\psi}_f$$

When ρ has only one non-zero eigenvalue (the case that there is a single event with probability one), it can be written as

$$\rho = \vec{\psi}(\vec{\psi})^\dagger$$

Then an active change of basis is given by

$$\mathcal{E}_t(\rho) = U_t\rho U_t^\dagger = U_t\vec{\psi}(\vec{\psi})^\dagger U_t^\dagger = (U_t\vec{\psi})(U_t\vec{\psi})^\dagger = \vec{\psi}_f(\vec{\psi}_f)^\dagger$$

Pure states

When $\rho = \vec{\psi}\vec{\psi}^\dagger$ show that the normalization condition on wave functions ($\|\psi\|_2 = 1$) follow from this decomposition of a pure state.

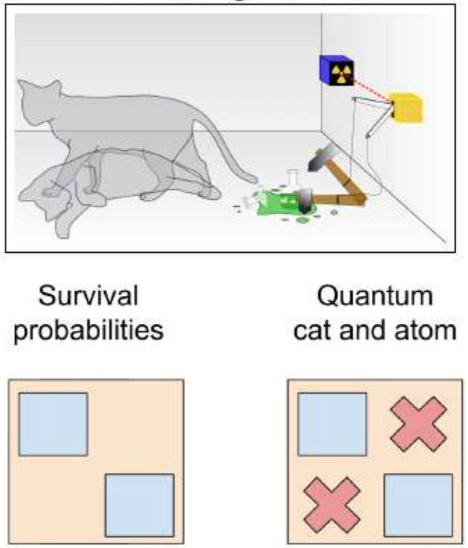
Outline

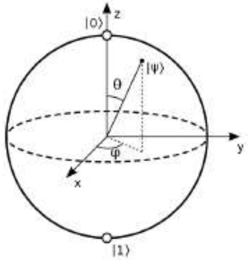
1. Probability
   - States
   - Kinematics: change of basis
   - Kinematics: transforming states
2. Quantum
   - States
   - Kinematics: change of basis
   - Kinematics: transforming states
3. Application: Quantum computing

Quantum Computing

Qubits

Schrödinger's cat





$$
\begin{aligned}
|\psi\rangle &= \alpha\,|0\rangle + \beta\,|1\rangle \\
&= \cos(\theta)\,|0\rangle + e^{-i\varphi}\sin(\theta)\,|1\rangle
\end{aligned}
$$

$$
\begin{aligned}
\mapsto \rho = |\psi\rangle\langle\psi| &= \begin{bmatrix} |\alpha|^2 & \alpha^*\beta \\ \beta^*\alpha & |\beta|^2 \end{bmatrix} \\
&= \begin{bmatrix} \cos^2(\theta) & \frac{e^{i\varphi}}{2}\sin(2\theta) \\ \frac{e^{-i\varphi}}{2}\sin(2\theta) & \sin^2(\theta) \end{bmatrix}
\end{aligned}
$$

In [76]:

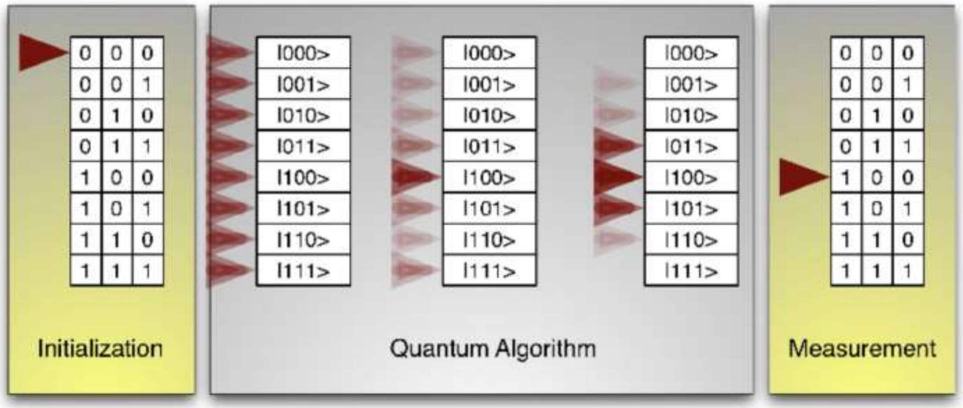
```
#random theta
q = np.random.rand() * 2 * np.pi
#random phi
f = np.random.rand() * 2 * np.pi

#the qubit wave function
psi = np.matrix( [[ np.cos(q) ] , [np.exp(-1j*f) * np.sin(q)] ] )

#quantum state (density matrix)
```

```
rho = np.matrix([[np.cos(q)**2 , .5 * np.exp(1j * f) * np.sin(2*q)],
       [.5 * np.exp(-1j * f) * np.sin(2*q), np.sin(q)**2]])

#standard quantum notation
ket0 = np.matrix( [[1],[0]])
ket1 = np.matrix( [[0],[1]])

#should be zero
print(psi @ psi.H - rho)
print(" ")
print(np.round(rho*10e4)/10e4 )
print(" ")
print((ket0 @ ket0.H) @ rho @ (ket0 @ ket0.H)
      + (ket1 @ ket1.H) @ rho @ (ket1 @ ket1.H))
```

```
[[0.+0.j 0.+0.j]
 [0.+0.j 0.+0.j]]

[[0.6508+0.j     0.0437-0.4747j]
 [0.0437+0.4747j 0.3492+0.j    ]]

[[0.6508+0.j 0.    +0.j]
 [0.    +0.j 0.3492+0.j]]
```

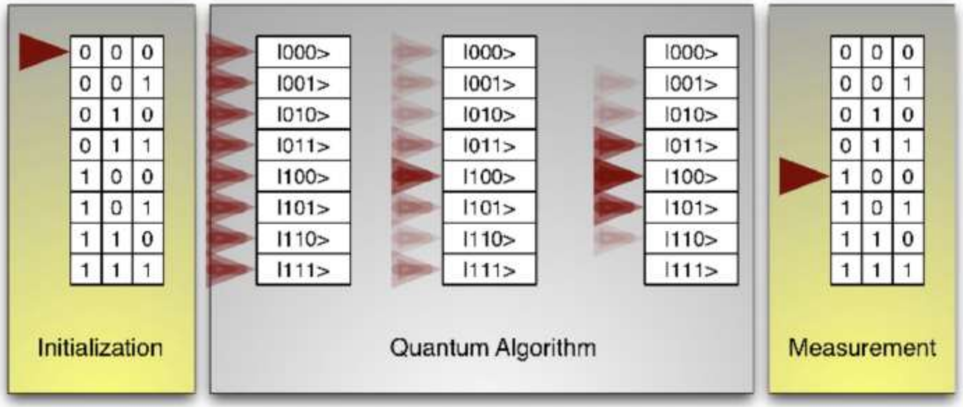
Quantum computing

Abstract parts of any quantum device



Quantum computing

Abstract parts of any quantum device



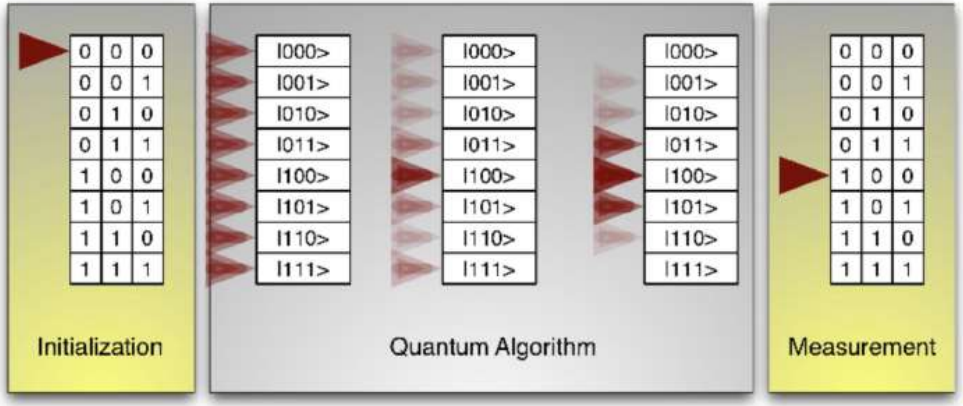
Projective measurements

$$\mathcal{E}_{\hat{e}}(\rho) = \sum_j (\hat{e}_j \hat{e}_j^\dagger) \rho (\hat{e}_j \hat{e}_j^\dagger)$$

This channel is need to measure and, if need, initialize the device

Quantum computing

Abstract parts of any quantum device



Active change of basis

$$\mathcal{E}_M(\rho) = \sum_{\alpha} w_\alpha U_\alpha \cdot [\rho] \cdot U_\alpha^\dagger$$

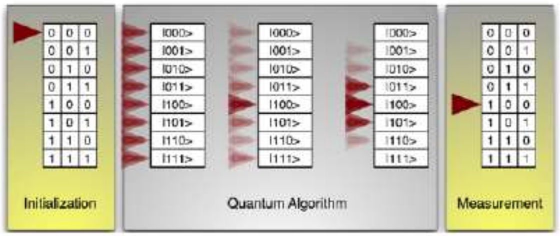
Ideal quantum computers implement, upon request, $U_{circuit}$ with high probability (i.e. $w_{circuit} \approx 1$)

Quantum supremacy

The point where quantum computers can do things that classical computers cannot, regardless of whether those tasks are useful

Alternate protocols

- FourierSampling
- BosonSampling
- IQP
- Random circuit sampling →



Article

Quantum supremacy using a programmable superconducting processor

Received: 22 July 2019
Accepted: 20 September 2019
Published online: 23 October 2019

The promise of quantum computers is that certain computational tasks might be executed exponentially faster on a quantum processor than on a classical processor[1]. A fundamental challenge is to build a high-fidelity processor capable of running quantum algorithms in an exponentially large computational space. Here we report the use of a processor with programmable superconducting qubits[2–7] to create quantum states on 53 qubits, corresponding to a computational state-space of dimension 2^{53} (about 10^{16}). Measurements from repeated experiments sample the resulting probability distribution, which we verify using classical simulations. Our Sycamore processor takes about 200 seconds to sample one instance of a quantum circuit a million times—our benchmarks currently indicate that the equivalent task for a state-of-the-art classical supercomputer would take approximately 10,000 years. This dramatic increase in speed compared to all known classical algorithms is an experimental realization of quantum supremacy[8–14] for this specific computational task, heralding a much-anticipated computing paradigm.

Quantum supremacy

- Relies on statistical arguments
- Requires implementing randomly selected $U_{circuit}$
- Sample from $\rho = \mathcal{E}_{circuit}[\rho(i)]$ with $\mathcal{E}_{circuit} \approx U_{circuit}(\bullet)U^{\dagger}_{circuit}$

- Note the approach I've given here is consistent with **any probability interpretation** of the state functions and of its measurements (frequentist, Bayesian, ontic, epistemic, etc.).
- Concept of correlations between subsystems in probability generalizes to the concept of **entanglement**
- **Superposition** in quantum mechanics is the same as superposition of music tones or of water waves. Typically, reaching superposition means increasing the fraction of non-zero elements in the density matrix but keeping the probability vector limited to a single non-zero entry.
- The **Heisenberg uncertainty principle** states the product of the uncertainty in the position, Δx, and of the uncertainty in the momentum, Δp, is always greater than some fixed number. This is a consequence of unitary the change of basis connecting position and momentum: the Fourier transform.

Welcome to the 2020 Quantum Winter School

Have Fun!

Welcome to Quantum:

- jdwhitfield.com (Research Group)
- qbraid.com (Learning Quantum Computing)