Dipartimento di Fisica Giuseppe Occhialini

Tesi di Laurea Triennale in Fisica

Esperimenti su singolo qubit:

implementazione e caratterizzazione di un quantum classifier

Relatore:
Dr. Giachero Andrea

Correlatore:
Dr. Carrazza Stefano

Candidato:
Galizzi Federico

Matricola:
839936

Anno Accademico 2020-2021

Sommario

Nel 1981, alla conferenza *"Physics of Computation Conference"*, Richard Feynman affermò la necessità di utilizzare computer basati sui principi della meccanica quantistica per poter indagare al meglio i fenomeni tipici del mondo atomico e subatomico. Computer quindi in grado di sfruttare effetti quali *entanglement*, interferenza e sovrapposizione di stati che l'informatica classica non utilizzava. Nacquero così i concetti di computer quantistico e *quantum computing.* Da allora, fisici, matematici, informatici e ingegneri hanno avuto a che fare con notevoli sfide da affrontare, tra le quali: le difficoltà di costruzione di sistemi così sofisticati e la stesura di algoritmi che possono effettivamente essere svolti da *device* quantistici. A quarant'anni da quella storica conferenza, IBM, che fu tra le promotrici dell'evento, offre un servizio cloud gratuito per imparare ad utilizzare computer quantistici, ad interagire con questi e per testare algoritmi che si basano sui principi del quantum computing. Il lavoro di tesi qui presentato è stato possibile grazie a questo servizio.
Il principale argomento di studio di questa tesi è il bit quantistico, comunemente chiamato *qubit*. Nel primo capitolo viene introdotto il concetto di qubit, presentandolo inizialmente come oggetto puramente matematico; astrazione con la quale i pionieri di questa nuova branca dell'informatica hanno lavorato. Tra essi si ricordano Peter Shor, David Deutsch e Richard Josza a cui si devono gli omonimi algoritmi verificati anni dopo la loro ideazione. Già su piano teorico, infatti, è possibile discutere circa cosa sia possibile e cosa impossibile fare con un qubit. In particolare, si presterà attenzione a come trarre vantaggio da un sistema che può essere in una qualsiasi combinazione di stati ma dal quale si può estrarre informazione classica solo attraverso un'operazione di misura; operazione che è assunta essere proiettiva da uno degli assiomi della meccanica quantistica. Proprio come per i bit tradizionali, gli unici risultati ottenibili da una singola misura applicata ad un qubit sono 0 e 1; indicati anche con |0> e |1>, in notazione di Dirac, per distinguerli dal caso classico.
A questa trattazione seguirà un'altra che più si avvicina alla realizzazione di un computer quantistico elencando e commentando i criteri di Davide DiVincenzo che, nel 2000, propose i requisiti minimi che un computer dovesse avere per essere adatto al quantum computing. Questi criteri, da allora, non sono stati alterati e negli anni successivi sono stati soddisfatti da diverse aziende (Google, IBM, Rigetti, D-Wave) con dispositivi che si basano su tecnologie come: difetti nei reticoli di diamanti, trappole di ioni, risonanza magnetica nucleare in molecole (*NMR*), quantum dots, fotoni e circuiti superconduttori. Di questi ultimi verrà fornita l'intera trattazione dell'Hamiltoniana del relativo sistema. Si è deciso di porre particolare attenzione ai qubit superconduttori in quanto su questa tecnologia si basano i computer di IBM, dunque anche il *backend* utilizzato qui per gli esperimenti. La quale scommette sui qubit superconduttori in quanto sono i più promettenti in materia di scalabilità, ossia sembrano essere i sistemi che permetteranno la costruzione di computer con maggior numero di qubit. Recentemente IBM ha annunciato che rilascerà nel 2023 un backend -*ibmq-condor*- costituito di 1121 qubit. Attualmente, il loro computer di punta ne ha 65.
A termine del capitolo 1 si introduce la *IBM Quantum Experience*, il servizio cloud già citato, e i due *framework* Qiskit e QiskitPulse che contengono le librerie necessarie per la scrittura e l'esecuzione di algoritmi su computer quantistici reali. Più precisamente, Qiskit serve per la costruzione di circuiti quantistici atti a svolgere i programmi tipici del quantum computing. QiskitPulse, invece, permette la creazione di programmi a livello di impulsi e consente quindi di avere un maggiore controllo sui qubit. Come usare questa ulteriore possibilità viene mostrato nel capitolo 2, con l'esperimento di

caratterizzazione di un qubit; il quale prevede l'utilizzo di impulsi nella banda delle microonde per trovarne la frequenza di risonanza attraverso tecniche di spettroscopia. Trovata questa, si procede con l'ottimizzazione dei parametri d'impulso necessari per implementare un X *gate* (l'analogo della porta NOT dell'informatica classica), grazie al quale si può far passare il qubit dallo stato fondamentale a quello eccitato e viceversa. Successivamente, si mostrano gli esperimenti necessari per la stima di T1 e T2 (i tempi caratteristici di rilassamento e decoerenza), due parametri fondamentali per soddisfare i criteri di DiVincenzo.

Il capitolo 3 si configura come lavoro di ottimizzazione dell'utilizzo di un qubit. Si studia quindi come poter far transire un qubit da uno stato ad un altro in maniera efficiente; un lavoro necessario in termini di affidabilità degli algoritmi di quantum computing.

Nel quarto capitolo si abbandona l'ambito dell'interazione con l'hardware a livello di impulsi per proporre un esperimento di implementazione di quello che potrebbe essere definito "*quantum classifier*". Sostanzialmente, viene descritto come sia possibile costruire un programma che, attraverso tecniche di *machine learning*, è in grado classificare dei dati in input in una classe precedentemente dichiarata e rappresentata da uno "stato-bersaglio" sulla sfera di Bloch. Compito non banale, in quanto i risultati dei circuiti quantistici restituiscono solo le ampiezze di probabilità di misurare |0> oppure |1>. Di questo programma vengono illustrate due versioni: una basata su un simulatore *ideale* e una basata su un simulatore di backend *reali*. Di quest'ultimo si discuterà l'inefficienza di classificazione raggiunta. L'esperimento è stato proposto con l'intento di mostrare i possibili vantaggi che il quantum computing può dare anche solo con un qubit, sebbene diversi libri di testo e articoli divulgativi individuano nell'entanglement e nell'interferenza (fenomeni che richiedono un numero plurale di qubit) i principali vantaggi dell'informazione quantistica nei confronti di quella classica.

Indice

Capitolo 1:

Qubit e IBM Quantum Experience

In questo primo capitolo sarà fornita un'approfondita descrizione di cosa sia un qubit e verranno descritte le possibilità fornite dall'IBM con il progetto di *IBM Quantum Experience*; in modo da avere tutti gli strumenti necessari per comprendere cosa sia stato svolto in questo lavoro di tesi.

1.1 Qubit

Come il *bit* è l'unità fondamentale della computazione classica, così il *quantum bit* (o semplicemente *qubit*) è l'unità fondamentale dell'informazione e della computazione quantistica [1]. Quest'analogia sarà la linea guida seguita per descrivere cosa sia un qubit. Il primo approccio per la descrizione di un qubit è puramente matematico, ignorando, per il momento, come questo possa essere costruito fisicamente. Si discute, dunque, un oggetto matematico.
L'oggetto in questione è quello che in meccanica quantistica viene chiamato *sistema a due livelli,* ossia una generica sovrapposizione (o combinazione lineare) di due stati normalizzata [2]. Usando la notazione di Dirac e indicando i due vettori (gli stati di cui prima) della base di uno spazio di Hilbert di dimensione 2 con |0> e |1>, qualsiasi combinazione di questi ammissibile è scrivibile come:

$$|\psi> = \alpha|0> + \beta|1>$$

dove ψ è lo stato in cui il sistema si trova, α e β coefficienti complessi che rispettano il vincolo di normalizzazione:

$$|\alpha|^2 + |\beta|^2 = 1$$

Per questo motivo ci si riferisce spesso a questi coefficienti come *ampiezze di probabilità*; in quanto, misurando un'osservabile A avente come autostati (o autovettori) |0> e |1> il loro modulo quadro indica proprio la probabilità di misurare l'uno o l'altro autostato.
Inoltre, dato che |0> e |1> costituiscono la base dello spazio considerato, questi autovettori godono di *ortonormalità*:

$$<0|0> = 1 \quad <1|1> = 1$$

$$<0|1> = 0 \quad <1|0> = 0$$

Per comodità successiva e semplicità di interpretazione, si introduce ora una rappresentazione più geometrica di un sistema a due livelli: la *sfera di Bloch*. La possibilità di usare un'interpretazione così geometrica è figlia diretta della condizione di normalizzazione. Si può infatti scrivere il generico stato $|\psi>$ come:

$$|\psi >= e^{i\gamma}(cos\frac{\theta}{2}|0 > +e^{i\varphi}sin\frac{\theta}{2}|1 >)$$

la quale, tenendo conto che un fattore di fase *overall* come è $e^{i\gamma}$ non può influenzare alcun modo le osservabili fisiche, si può semplificare come segue:

$$|\psi >= cos\frac{\theta}{2}|0 > +e^{i\varphi}sin\frac{\theta}{2}|1 >$$

dove $0 \leq \theta < \pi$ e $0 \leq \varphi < 2\pi$. Proprio questi due parametri reali definiscono un punto nella rappresentazione grafica (fig 1.1) della sfera di Bloch, se si considera θ che il vettore-stato forma con l'asse delle Z e φ l'angolo azimutale.

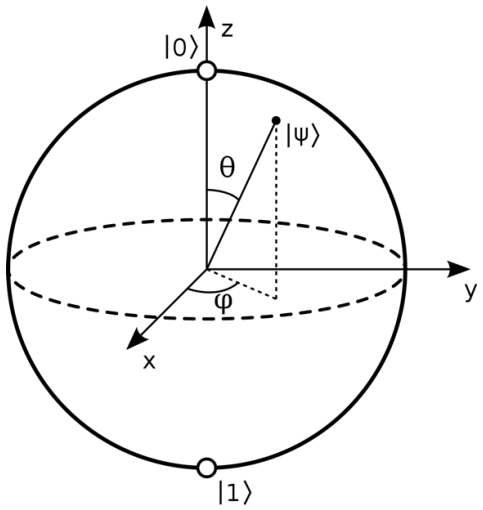


Fig 1.1: Rappresentazione della sfera di Bloch

È ora il momento di discutere di quanta informazione può essere contenuta in un qubit. Il quesito non è banale; è anzi cruciale per capire se il quantum computing possa davvero offrire vantaggi rispetto all'informatica classica. Di fatti, sebbene ci sia un'infinità di punti sulla superficie della sfera unitaria, il risultato di un'osservazione può restituire solamente |0> o |1> con le probabilità discusse prima. Non solo, l'operazione di misura è proiettiva, quindi la sovrapposizione di stati viene istantaneamente persa quando questa è applicata! In meccanica quantistica si parla di "collasso della funzione d'onda", un fenomeno non ancora spiegato, ma postulato, e irreversibile. Sarà ora evidente anche al lettore meno afferrato in meccanica quantistica perché la stesura di algoritmi quantistici necessitino di una logica completamente diversa da quella dell'informazione classica.
Per completezza, ma argomento che esula dalle intenzioni di questo lavoro di tesi, si vuole dire che sì, il quantum computing ha grossi vantaggi rispetto alle tecniche tradizionali in termini di *costi computazionali*; anche in virtù del fatto che sfrutta altri fenomeni della meccanica quantistica che emergono in presenza di più sistemi (qubit): *entanglement* e interferenza.

1.2 Operazioni su singolo qubit

Si continua in questo paragrafo un parallelo tra informazione classica e quantistica per esaltare le maggiori possibilità che questa seconda comporta.
Nell'informatica tradizionale, l'unica porta logica non banale che si può applicare ad un singolo bit è la porta NOT che opera nel modo che segue:

$$0 \rightarrow 1$$
$$1 \rightarrow 0$$

Il compito che assolve questa porta è sostanzialmente quello di scambiare 0 con 1, e viceversa. Non ci sono altre operazioni non banali (e.g. 1→1) che si possono fare su un singolo bit. Il motivo è semplice: si hanno a disposizione due soli stati.
E nel quantum computing? Nel quantum computing l'infinità di stati che un qubit può assumere comporta altrettante operazioni applicabili ad esso.
Chiaramente esistono delle analoghe porte logiche, chiamati più comunemente *gate* in questo contesto. Si hanno infatti l'operazione d'identità

$$|0> \rightarrow |0>$$
$$|1> \rightarrow |1>$$

e l'analogo NOT gate

$$|0> \rightarrow |1>$$
$$|1> \rightarrow |0>$$

È necessario qui far notare che questi gate sono in realtà molto più generali. Infatti, questi possono agire su qualsiasi stato in cui si trova il qubit, dunque l'identità è scrivibile come:

$$|\psi> \rightarrow |\psi>$$

Per apprezzare come agisce il NOT gate, chiamato anche X gate per i motivi che seguono, è conveniente introdurre la notazione matriciale secondo la quale

$$|0> = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \qquad |1> = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

In questa notazione l'X gate è rappresentato dalla matrice $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$. Per cui, nel caso di un generico stato $\alpha|0> + \beta|1>$, esso agisce così:

$$X \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \equiv \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} \beta \\ \alpha \end{pmatrix}$$

Ora, prima di mostrare altri esempi di gate, si vuole rispondere ad una domanda che qui sorge spontanea: per accedere ad uno qualsiasi degli infiniti punti/stati della sfera di Bloch, quanti gate servono? Infiniti? Fortunatamente, il numero di gate necessari è finito. Si può provare l'esistenza di *set universale di gate*, ossia un numero finito di gate che permettono di approssimare con precisione

asintotica qualsiasi operazione si intenda applicare sullo stato. In altri termini: dato un punto sulla sfera di Bloch, si può raggiungere qualsiasi altro punto con una combinazione finita dei gate appartenenti al set universale. Più in generale, nel caso di sistemi a più qubit, si vuole saper implementare "qualsiasi trasformazione unitaria" il che include anche l'*entanglement*; ma questo non riguarda gli esperimenti a singolo qubit.

In seguito, una lista di gate comunemente usati nel quantum computing:

$$Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \quad Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \quad H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad P = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\varphi} \end{bmatrix} \quad S = \begin{bmatrix} 1 & 0 \\ 0 & e^{-i\frac{\pi}{2}} \end{bmatrix}$$

Tra questi, Y e Z inducono una rotazione di π radianti attorno al rispettivo asse; H è noto in letteratura come *Hadamard gate*, I è l'identità (non apporta alcuna modifica allo stato del qubit), P è il *phase gate*, dunque altera la fase relativa tra |0> e |1> dello stato attuale e S non è che un caso particolare del P gate.
Tralasciando momentaneamente il fatto che l'esistenza di un set universale è fondamentale per il quantum computing, in quanto il più delle volte si è in grado implementare fisicamente solo certi gate, idealmente, qualsiasi operazione unitaria su singolo qubit può essere rappresentata da un gate a tre parametri: l'U3 gate; la cui forma matriciale risulta essere:

$$U3(\theta, \varphi, \lambda) = \begin{pmatrix} cos\frac{\theta}{2} & -e^{i\lambda} sin\frac{\theta}{2} \\ e^{i\varphi} sin\frac{\theta}{2} & e^{i(\varphi+\lambda)} cos\frac{\theta}{2} \end{pmatrix}$$

1.3 Realizzazione di un qubit

In questo paragrafo si illustra come vengono realizzati fisicamente i qubit e come vengono controllati; un passaggio dal teorico al pratico che storicamente ha richiesto decine di anni (Feynman propose il concetto di *quantum computer* già nel 1981!). Non verranno descritte le tecniche di costruzione di un qubit, in quanto non si intende approfondire l'aspetto ingegneristico di questo arduo compito, si illustreranno "semplicemente" dei principi guida.
La stesura dei criteri che deve soddisfare un sistema fisico per poter essere considerato l'*hardware* di un computer quantistico la si deve a Davide DiVicenzo [3]; il quale, nel 2000, propose queste cinque condizioni necessarie:

1. Possibilità di identificare qubit ben definiti ed aumentarli di numero. Questo criterio richiede una solida rappresentazione dell'informazione.
2. Capacità di inizializzare lo stato del sistema: portare con estrema efficacia i qubit in un determinato stato iniziale e, solo dopo, eseguire un calcolo.
3. Tempi di coerenza sufficientemente lunghi.
4. Esistenza di una classe universale di porte logiche quantistiche (gate) per il controllo preciso dei qubit; sostanzialmente, essere in grado di implementare tutti i gate appartenenti ad un set universale.
5. Effettuare una misura per ottenere il risultato del calcolo eseguito e trasmettere l'informazione quantistica.

Ora un breve e doveroso commento a questi criteri.

"Possibilità di identificare qubit ben definiti ed aumentarli di numero": riprendendo quanto discusso nei paragrafi precedenti, un qubit è un sistema a due livelli ed ogni sistema fisico scelto per realizzarlo deve assicurarci il fatto che questo non esca mai dal sottospazio (di Hilbert) di questi due livelli. Per quanto già questo sia di difficile costruzione, "aumentarli di numero" è la parte di questo criterio che attualmente rappresenta la sfida più ardua da superare.

"Capacità di inizializzare lo stato del sistema": questa richiesta, piuttosto semplice nell'informazione classica (è equivalente a richiedere di essere in grado di settare un bit a 0), non è banale nel mondo quantistico. Si noti che la richiesta è quella di saper inizializzare *uno* stato iniziale degli infiniti che costituiscono la sfera di Bloch. Dal punto di vista teorico non esiste alcuno stato privilegiato, dal punto di vista pratico, generalmente, si preferisce inizializzare un qubit allo stato fondamentale |0>. D'altro canto, gli stessi assiomi della meccanica quantistica vengono in soccorso in questo: dato che la misura è proiettiva, ogni volta che misuriamo lo stato di un qubit sulla base [|0> ; |1>] e otteniamo |0> si è certi che immediatamente dopo la misura lo stato fisico del sistema sia |0>. Attenzione: saper effettuare una misura è la richiesta del quinto criterio.

"Tempi di coerenza sufficientemente lunghi": questa richiesta, non sempre incluso tra i criteri di DiVincenzo nei libri testo, rispecchia un *leitmotiv* della fisica: la realtà presenta dei "difetti" rispetto al mondo ideale. I difetti che non possiamo trascurare, in questo caso, sono la non stazionarietà del livello eccitato |1> (tende a decadere allo stato di *ground* |0>) e l'interazione del sistema con l'ambiente. I tempi di decoerenza di un qubit saranno oggetto di discussione del capitolo 2 come parametro fondamentale per la caratterizzazione; l'espressione "sufficientemente lunghi" va intesa in relazione alla durata dei gate applicati al qubit e al numero di operazioni necessarie per svolgere un determinato algoritmo.

"Esistenza di una classe universale di porte logiche quantistiche": sebbene l'esistenza di un set universale di gate sia assicurato da un teorema, l'implementazione dei gate appartenenti a questo non va considerata un *task* semplice da affrontare anche in virtù del fatto che gli algoritmi di quantum computing richiedono spesso un'elevata precisione dei gate.

"Effettuare una misurazione": si richiede di essere in grado di trasformare un'informazione quantistica in informazione classica. Non solo, bisogna essere in grado di farlo con un'adeguata accuratezza.

I criteri di DiVincenzo non citano alcuna tecnica di costruzione e nemmeno una modalità di controllo di un qubit. La mancanza di questi vincoli permette di teorizzare qualsiasi algoritmo quantistico senza che effettivamente esista un computer quantistico, di fatto alcuni algoritmi precedono la prima realizzazione di un computer di questo genere, e inoltre lascia libertà nella scelta di sistemi fisici adatti a questi scopi.
Negli anni, son stati creati qubit con fotoni, "ioni intrappolati" (preferibile l'espressione inglese *ion in traps*), circuiti superconduttori, difetti nei reticoli di diamante (NV diamonds), quantum dots, risonanza magnetica nucleare in molecole (*NMR in molecules*) ecc.
Di questi si potrebbe discutere diffusamente di pro e contro, ma ai fini di questa tesi si intende fornire una semplice e schematica (fig1.2) classificazione in termini di tempi di coerenza e velocità prima di

trattare singolarmente i qubit realizzati con circuiti superconduttori; non perché si vuole affermare che siano i migliori, ma semplicemente perché sono quelli usati da IBM e, di conseguenza, quelli usati nella parte sperimentale di questo lavoro.

Fig 1.2: Classificazione presa da [4]. In questo caso,
"velocità" indica la durata di una singola operazione (gate).

Attualmente, i leader mondiali nella costruzione di computer quantistici basati su architettura di circuiti superconduttori sono Google e IBM. Google, con *Sycamore*, vanta la costruzione di un computer a 53 qubit [5]. Con l'impiego di questo backend, l'azienda statunitense ha annunciato di aver raggiunto la *quantum supremacy* nell'ottobre del 2019 [6]. Un numero maggiore di qubit è stato raggiunto da IBM con due diversi backend, *ibmq-manhattan* e *ibmq-brooklyn*, entrambi computer da 65 qubit e *quantum volume* pari a 32. Questo parametro, il quantum volume, è una possibile metrica per misurare le prestazioni di un computer quantistico. È un numero intero stimato in base alla complessità dei circuiti che un computer può effettivamente implementare. Tendenzialmente, il suo valore cresce col numero dei qubit, sebbene, in realtà, il suo calcolo si basa sui valori di fedeltà delle operazioni, sul grado di connettività tra i diversi qubit, sul set di gate accuratamente calibrati e sull'efficacia di mappatura del circuito [7]. Di fatti, IBM dichiara di aver raggiunto valori di quantum volume maggiori con computer a 27 qubit, quali *ibmq-montreal* e *ibmq-dublin*, per i quali è stato stimato essere pari a 128 e 64, rispettivamente. Invece, il backend utilizzato per questa tesi, *ibmq-armonk*, è a singolo qubit e ha quantum volume pari a 1.
Circa il confronto tra computer quantistici che si basano su tecnologie diverse, si riportano valori tipici di velocità e tempi di decoerenza. Quelli basati su ion traps hanno valori tipici di durata di singolo gate tra i 0,1μs e i 5μs con valori che crescono per operazioni su più qubit (50-3000μs) [8]; per questo tipo di qubit si è raggiunto un tempo di coerenza di circa 6000s, oltre un'ora [9]. Per i qubit superconduttori, invece, si attestano valori nettamente inferiori sia per il tempo di implementazione di un gate, sia per il tempo di coerenza. Rispettivamente, valori tipici possono essere di decine di nanosecondi e centinatia di microsecondi; che comunque consentono di poter applicare un numero di operazioni nell'ordine di $10^3 - 10^4$; quelle richieste dai criteri di DiVicenzo.
Uno dei motivi per cui multinazionali come IBM e Google, per l'appunto, reputino i computer quantistici a superconduttori preferibili è che questi promettono maggiori possibilità in termini di scalabilità. Sono infatti gli unici ad aver raggiunto le decine di qubit e, al tempo stesso, avere un'architettura che permetta l'implementazione di sistemi ancora più complessi. Caratteristica

estremamente interessante, dato che la potenza di calcolo di questi computer è esponenziale in funzione del numero di qubit. Dal blog di IBM si legge che sono in corso i lavori per la costruzione di un device da 1121 qubit, *ibmq-condor*, il termine dei quali è previsto nel 2023 [10].

1.4 Qubit superconduttori

Come annunciato, si approfondisce qui il caso di qubit realizzati con circuiti superconduttori [11]. L'idea è quella di realizzare un oscillatore armonico con un circuito LC, dove compaiono due elementi di circuito: l'induttore (di induttanza L) e il capacitore (di capacità C). Come noto, le energie contenute in questi due elementi sono:

$$E_C = \frac{1}{2}CV^2 = \frac{Q^2}{2C} \qquad E_L = \frac{1}{2}LI^2 = \frac{\phi^2}{2C}$$

dove Q=CV è la carica del capacitore e ϕ=LI è il flusso di campo magnetico nell'induttore. Segue che l'Hamiltoniana classica è la somma di questi due termini:

$$H_{Cl} = \frac{Q^2}{2C} + \frac{\phi^2}{2C} = \frac{Q^2}{2C} + \frac{1}{2}C\omega_0^2\phi^2$$

in particolare, nella seconda formulazione è stato introdotto il parametro $\omega_0 = 1/\sqrt{LC}$ per potere identificare più facilmente un Hamiltoniana di "massa" C, momento Q, "posizione" ϕ e frequenza ω_0. In questo modo si può riscrivere H_{Cl} in termini di operatori di *creazione* $(a^\dagger)$ e *distruzione* (a) come noto dalla meccanica quantistica. Sostituiamo quindi Q e ϕ coi rispettivi operatori in termini di a e $a^\dagger$.

$$Q \to \hat{Q} = i\sqrt{\frac{\hbar}{2Z_0}}(a - a^\dagger) \qquad \phi \to \hat{\phi} = i\sqrt{\frac{\hbar Z_0}{2}}(a + a^\dagger)$$

dove $Z_0 = \sqrt{L/C}$ è l'impedenza. Si ricorda che valgono le regole di commutazione:

$$[a, a^\dagger] = \mathbb{I} \qquad [\hat{Q}, \hat{\phi}] = i\,\hbar$$

Si è ora in grado di riscrivere:

$$\hat{H}_{LC} = \hbar\omega_0(aa^\dagger + \tfrac{1}{2}) \quad \hat{H}_{LC}|n> = E_n|n>$$

ora |n> sono gli autostati e $E_n = \hbar\omega_0(n + 1/2)$ gli autovalori dell'Hamiltoniana scritta. Siccome la differenza di energia di un livello energetico e il successivo è sempre pari a $\hbar\omega_0$, è impossibile selezionare solo due stati (condizione necessaria per il qubit). Si sceglie, dunque, di introdurre dei termini non-lineari così da rompere la regolarità dei "salti" di energia e la ΔE_{0-1} è unica.
Per ottenere questo risultato, il circuito è reso anarmonico sostituendo l'induttore L con una giunzione Josephson. Una giunzione Josephson consiste di due superconduttori separati da un isolante. Il suo funzionamento si basa poi sull'effetto tunnel della coppia di Cooper attraverso lo strato isolante.

Rimandando all'Appendice la trattazione dell'Hamiltoniana di questo sistema e i calcoli necessari per ottenere i nuovi livelli energetici (usando la teoria delle perturbazioni) si illustrano in fig 1.3 gli schemi delle giunzioni Josephson e i grafici dei livelli energetici del sistema armonico (non perturbato) e del sistema anarmonico (perturbato).

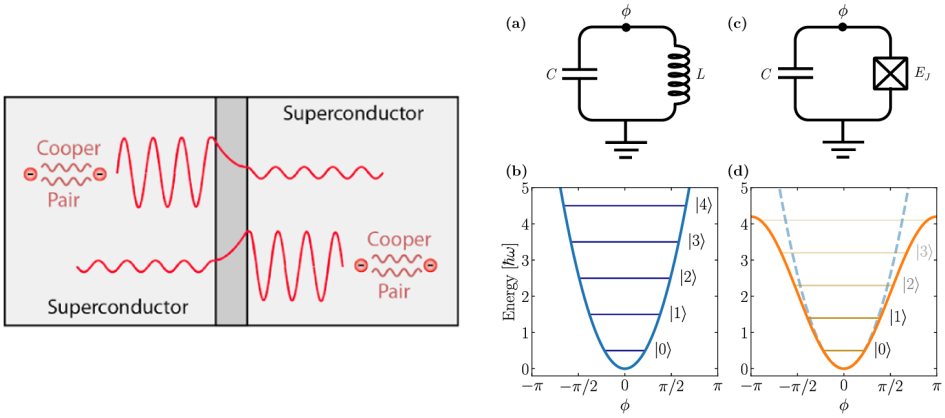


Fig 1.3: A destra un'idea di funzionamento della giunzione Josephson in cui si può apprezzare una rappresentazione qualitativa dell'effetto tunnel. A sinistra gli schemi di un circuito LC (oscillatore armonico) e di un circuito con SQUID in serie ad un capacitore e i rispettivi livelli energetici.

1.5 IBM Quantum Experience

IBM Quantum Experience è un servizio cloud messo a disposizione da IBM dal quale è possibile accedere a computer quantistici. Con un account "base" e gratuito è possibile utilizzare hardware composti di 1 o 5 qubit oppure sfruttare dei simulatori di sistemi formati da 32 fino a 5000 qubit; ai primi non si ha accesso diretto e immediato: bisogna inviare al cloud l'algoritmo che si intende eseguire, la richiesta viene messa in coda alle altre di altri utenti e solo successivamente viene eseguita; per i simulatori, invece, non c'è alcuna lista d'attesa [12]. I qubit usati da IBM sono superconduttori di tipo transmon, con i quali si può interagire grazie a due *framework* open source: Qiskit e Qiskit Pulse; entrambi in linguaggio Python.
Con Qiskit si può creare qualsiasi circuito quantistico usando un'interfaccia grafica (*composer*) o direttamente programmando in Python. Il composer è dotato di cinque righe, rappresentati 5 qubit diversi inizializzati allo stato fondamentale |0> ai quali si possono applicare diversi gate scegliendo da una banda laterale. Da questa banda si può importare anche l'operazione di misura del qubit per salvare l'informazione quantistica su un bit classico e ottenere così dell'informazione classica utilizzabile. Solitamente, il composer è utilizzato più per scopo didattico in quanto non offre tutte le possibilità che si hanno programmando con le librerie di Qiskit. "Per scopo didattico" anche perché l'interfaccia grafica mostra come si modifica lo stato del qubit sulla sfera di Bloch ogni qualvolta si aggiunge un gate e dà una stima percentuale delle misure di |0> o |1> che si otterrebbero facendo girare il circuito creato su un computer quantistico. Questo è sicuramente un comodo approccio al mondo del quantum computing per chi non è esperto di meccanica quantistica o non ha affinità con gli operatori. Inoltre, grazie al fatto che si possono far girare i programmi-circuiti realizzati sia su un simulatore che su un vero computer quantistico, ci si può far un'idea della discrepanza che c'è tra "computer ideale" (rappresentato dal simulatore, il quale assume la totale assenza di errori e rumore elettronico) e "computer reale" osservando i risultati ottenuti dall'hardware quantistico.

Qiskit Pulse, invece, permette di lavorare con i quantum computer ad un livello più vicino all'hardware. Non va dimenticato che per implementare fisicamente un circuito quantistico utilizzando dei qubit superconduttori richiede saper "tradurre" delle istruzioni informatiche (esempio banale: applicare un X gate) in impulsi di microonde. Qiskit Pulse permette proprio questo: conoscere la corrispondenza gate-impulsi. Nel capitolo successivo, che riguarda la caratterizzazione di un qubit, verrà descritto un programma che usa solamente le librerie di Qiskit Pulse e si mostrerà come implementare sia l'X gate, ottimizzando un impulso gaussiano per passare da |0> a |1> e viceversa, sia l'Hadamard gate; col quale, si ricorda si passa dallo stato |0> alla combinazione lineare $\frac{|0>+|1>}{\sqrt{2}}$.
Vale la pena qui introdurre il *backend* utilizzato in questo lavoro di tesi: ibmq_armonk. È il più "piccolo" dei computer messi a disposizione da IBM [13], consta di un solo qubit e deve il suo nome alla città in cui ha sede IBM. Le caratteristiche principali di questo computer sono:
- Tipo di processore: Canary r1.2
- Base di gate: I, RZ, SX, X
- Errore di readout medio: 3.86e-2
- T1 medio: 211.96 μs
- T2 medio: 223.56 μs
- Frequenza di risonanza: 4.972 GHz

Il lavoro di calibrazione del qubit viene svolto giornalmente da IBM, aggiornando così i valori appena mostrati i quali possono piccole variazioni dovute ad alterazioni delle condizioni dell'ambiente di lavoro (ad esempio un aumento della temperatura interna al criostato in cui sono posti i qubit). Una calibrazione frequente permette di avere dei gate sempre ottimizzati alle condizioni attuali del qubit e, dunque, ad avere le migliori prestazioni ottenibili dallo stesso. Un modo per poter svolgere questo lavoro automaticamente con Qiskit Pulse è mostrato nel capitolo che segue.

Capitolo 2:

Caratterizzazione di un qubit

L'esperimento di caratterizzazione di un qubit qui proposto prevede l'utilizzo di un programma scritto in linguaggio Python. Tale programma utilizza le librerie di Qiskit [14] e QiskitPulse [15] per controllare un hardware quantistico messo a disposizione dall'IBM sulla piattaforma IBM-quantum-lab; in particolare, è stato utilizzato il backend *ibmq_armonk*.
Senza entrare nel dettaglio di ogni singola funzione implementata, viene di seguito descritta la procedura di caratterizzazione; il programma completo è riportato in appendice. Questo esperimento assume solo la conoscenza di una precedente stima di frequenza di risonanza; nel caso non la si conoscesse, cambierebbe solo la prima parte di esperimento: gli impulsi mandati in ingresso al qubit dovrebbero ricoprire un grande intervallo di frequenze entro il quale è ragionevole pensare che si trovi la risonanza, questo intervallo potrebbe essere scelto in base alle caratteristiche di costruzione del qubit stesso. Valori tipici si attestano sui 4-5 GHz.
Ogni valore numerico che compare in questo capitolo non ha alcuna validità generale; in quanto, queste procedure di calibrazione devono essere svolte quotidianamente su ogni qubit di qualsiasi device quantistico.

2.1 Frequenza di risonanza

Il primo step necessario per la calibrazione è trovare la frequenza di risonanza del qubit e dunque la differenza di energia tra lo stato fondamentale |0> e il primo stato eccitato |1>; gli unici due stati che si assumono accessibili in questo tipo di esperimento. Per farlo, si inviano impulsi di microonde in un range di frequenze centrato alla frequenza precedentemente stimata. Per ogni impulso in ingresso, si misura il segnale in uscita da qubit aspettandosi di osservare un picco: è proprio quest'osservazione che ci permette di stimare la frequenza di risonanza. La fig 2.1 mostra l'andamento del segnale letto in uscita in funzione delle frequenze degli impulsi inviati.

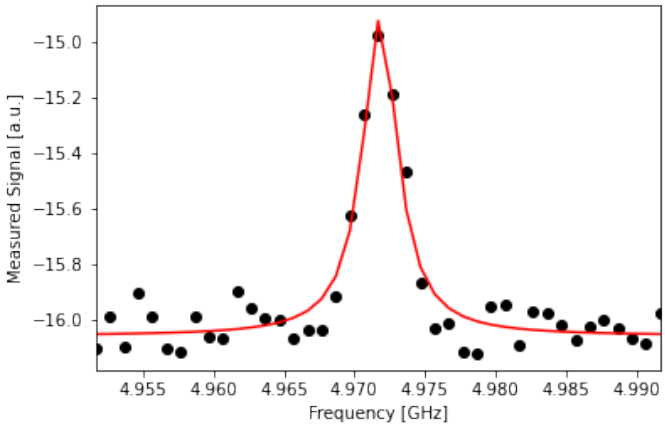


Fig 2.1: il grafico mostra un picco molto pronunciato giusto al centro del range di frequenze inviate in ingresso, l'ascissa di questo picco rappresenta la prima stima di frequenza di risonanza.

I dati sono stati fittati con una Lorentziana di cui ci interessa il valore centrale, il quale indica la frequenza di risonanza. Valore stimato: 4.972GHz.
Il valore ottenuto va considerato una prima approssimazione: una tecnica più performante sarà descritta in seguito e si baserà su questo risultato.

2.2 Calibrazione di impulsi

Altro aspetto fondamentale per la caratterizzazione di un qubit è la calibrazione degli impulsi che implementano fisicamente i gate che si applicano in un circuito quantistico. Si pensi ai gate come le operazioni elementari del quantum computing.
In questa sezione vengono descritte le calibrazioni degli impulsi necessari per ottenere i gate π e π/2; nomi che si riferiscono alle ampiezze degli angoli di rotazione attorno all'asse delle X che questi gate apportano allo stato del nostro qubit (riferendosi alla sfera di Bloch). La procedura qui seguita è quella dell'esperimento noto in letteratura come *Rabi experiment* [16]; per la quale si scelgono a priori la forma dell'impulso e la sua durata: in questo caso sono stati scelti impulsi gaussiani di larghezza - ovvero la *sigma* - 0,075μs (mentre la frequenza utilizzata, chiaramente, è quella di risonanza trovata precedentemente). Si vuole specificare che, in questo esperimento, si è abbastanza liberi di scegliere la forma dell'impulso, sebbene per qubit superconduttori è preferibile scegliere una forma rappresentabile da una funzione il più possibile liscia (o *smooth*) [17] [18].
Si descrive ora l'esperimento. Al qubit in stato fondamentale vengono mandati in ingresso degli impulsi di durata costante e ampiezza via via crescente. Per ogni valore di ampiezza si inviano 1000 impulsi e si esegue una misura di energia al qubit. La frazione di volte che si ottiene |1> permette di identificare lo stato del qubit che in media si ottiene con questo valore di ampiezza. Si pensi di nuovo al qubit come un sistema a due livelli, il cui generico stato è rappresentabile come $\alpha|0> + \beta|1>$; considerando la condizione di normalizzazione $|\alpha|^2 + |\beta|^2 = 1$ possiamo ottenere i due coefficienti a meno della una fase relativa. Quello che si osserva, al variare dell'ampiezza degli impulsi inviata, sono delle oscillazioni periodiche proprio dei moduli quadrati di α e β. Questo fenomeno prende il nome di Oscillazioni di Rabi; la cui trattazione teorica è fornita in appendice, qui eliminata per semplicità espositiva.

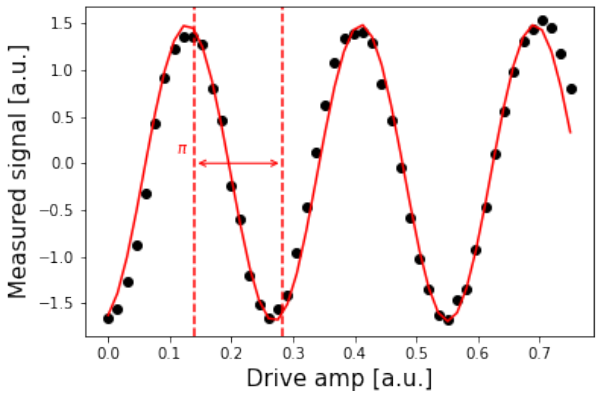


Fig 2.2 Le oscillazoni di Rabi

Dal fit di questi dati - fig 2.2 - con una sinusoide è possibile ottenere l'ampiezza necessaria per il gate π, rappresentata dal semiperiodo delle oscillazioni.

Forma funzionale usata:

$$A\cos\left(\frac{2\pi x}{T} - \varphi\right) + B$$

Dove A è una costante, x è il valore variabile dell'ampiezza dell'impulso inviato, T è il parametro rappresentante il periodo in ampiezza e sia φ -fase iniziale- che B sono altre due costanti
Si spiega ora il motivo per cui la durata dell'impulso sia stata tenuta costante. Si ricordi che l'obbiettivo che ci si è posti è esclusivamente l'implementazione di un gate π accurato: l'importante è essere in grado di ruotare di un angolo π attorno all'asse delle X lo stato sulla sfera di Bloch e lasciar libero solo il parametro d'ampiezza è sufficiente per ottenere questo risultato. È pur vero che la durata del gate potrebbe avere ripercussioni per quanto interessa il quantum computing; per il quale, dati i tipici tempi di decoerenza di un sistema fisico (qubit), è molto vantaggioso avere gate brevi in durata in modo tale da poter implementare algoritmi che richiedano un numero maggiore di operazioni. Questo aspetto è discusso nel capitolo 3 come punto fondamentale del lavoro di ottimizzazione. Si è ora in grado di promuovere lo stato |0> allo stato eccitato |1> con ottima accuratezza.

2.3 Discriminatore di |0> e |1>

Dal momento in cui si è in grado di implementare l'X gate (il nome che la letteratura dà al gate fin qui chiamato π) si intende ora mostrare come distinguere lo stato di ground e lo stato eccitato con una misura.
L'esperimento è assai elementare. Si creano due semplici circuiti quantistici: il primo è costituito da una sola operazione di misura del qubit, il secondo, invece, prevede di applicare l'X gate appena implementato e successivamente effettuare la misura. Si può quindi assumere che gli stati sui quali si effettuano le misure siano o |0> oppure |1> e non sovrapposizioni di questi.
In figura (fig 2.3) sono rappresentate le sequenze di impulsi inviati al qubit per i rispettivi circuiti.

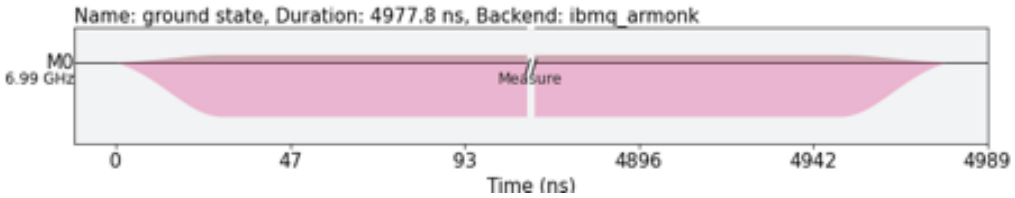


Fig 2.3: sola misura dello stato quantistico

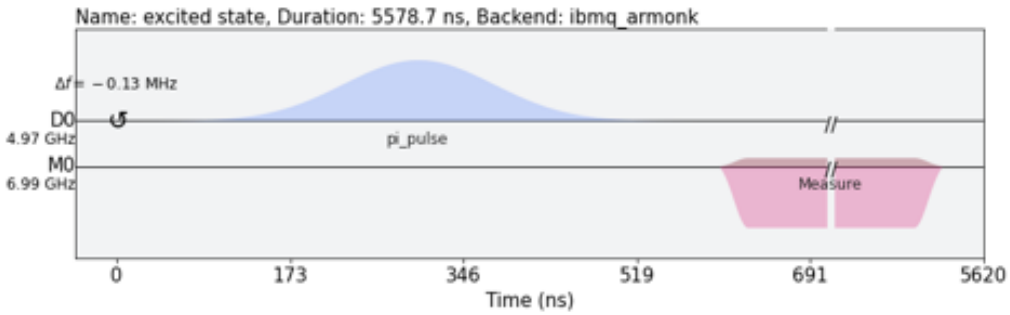
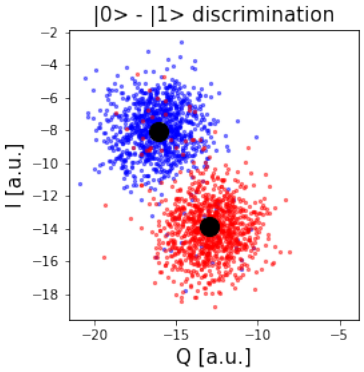


Fig 2.3: Applicazione di un impulso π seguito da immediata misura

I risultati delle misure, che anch'esse sono impulsi alla frequenza di risonanza del qubit, vengono interpretati come punti nel piano complesso (parte reale; parte immaginaria) con unità arbitrarie proprie delle librerie di Qiskit.



Si può facilmente notare come vadano a configurarsi due raggruppamenti ben definiti. I punti blu sono i risultati delle misure del programma di preparazione dello stato fondamentale (la sola misura) mentre i punti rossi sono quelli del programma con un X gate seguito da misura.
Da questi risultati si può costruire una funzione per discriminare i due stati. Si procede trovando il punto medio di uno e dell'altro cluster (evidenziati in nero nella figura), si dichiara una funzione che calcola la distanza di un generico punto da l'uno o dall'altro centro cosicché, per programmi o circuiti successivi, ogni risultato di misura sarà interpretato come |0> se il rispettivo punto sarà più vicino al punto medio del raggruppamento blu e |1> se altrimenti.
Si ricorda che in meccanica quantistica la misura è proiettiva, non si può dunque assolutamente pensare che i punti lontani dai "centri" possano essere interpretati come stati sovrapposti e che quindi ci possa essere un modo per ottenere informazioni sui coefficienti α e β dello stato in cui era il qubit prima della misura. Anzi, il fatto che si osservano due cluster, piuttosto che due soli punti nel piano, si deve al rumore o all'imperfezioni proprie del qubit e dell'apparato di misurazione.

2.4 Tempo di rilassamento (T1)

Il tempo di rilassamento di un qubit è un parametro fondamentale nella valutazione delle prestazioni dello stesso; in particolare, è importante che questo tempo caratteristico sia molto maggiore rispetto alla durata tipica di un'operazione sul qubit (gate). Comunemente chiamato anche T1, il tempo di rilassamento è definito come l'intervallo di tempo necessario affinché la probabilità di misurare |1> per un qubit preparato nel suo stato eccitato a t = 0 si riduca da 1 a 1/e. La definizione di T1 è in completa analogia con la definizione di tempi di decadimento degli isotopi radioattivi (o delle eccitazioni atomiche) in quanto la natura del fenomeno descritto è esattamente la stessa: stocastica. Come l'istante in cui decade un isotopo è assolutamente imprevedibile, la stessa cosa vale per il passaggio dallo stato |1> allo stato |0> per un qubit se su questo non agisce alcun gate ed è, dunque, lasciato libero di interagire con l'ambiente. Si tratta in questo caso di esperimenti che richiedono molta statistica per poter ottenere delle stime affidabili.
Come si potrà poi constatare, il procedimento qui descritto utilizza tutto ciò che è stato implementato finora: l'X gate propriamente calibrato e la funzione *discriminator*. Di fatto, rispetto a quanto fatto finora c'è solo una novità: oltre ai gate (implementati con gli impulsi trovati) e all'operazione di misura, ci avvaliamo anche di tempi di ritardo (*delay time*) tra gli uni e gli altri. Tempi che possiamo inserire nel circuito quantistico in maniera del tutto arbitraria grazie alle librerie di Qiskit.

L'esperimento prevede tre operazioni: applicare un π-pulse, inserire un ritardo e applicare una misura. Questo è stato iterato per 250 volte per ognuno dei 70 valori di delay time diversi (da $1\mu s$ a $450\mu s$ a step di $6{,}5\mu s$).

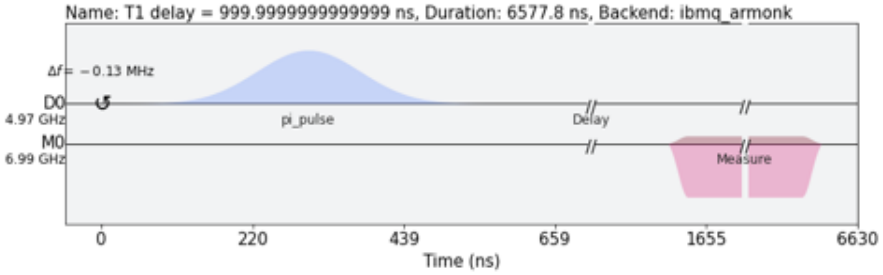


Fig 2.4: rappresentazione grafica dell'esperimento

Con questi dati si costruisce il grafico (fig 2.5) frazione di qubit nello stato |1> in funzione del delay inserito e si fittano i dati con un esponenziale decrescente della forma:

$$Ae^{-t/T1} + B$$

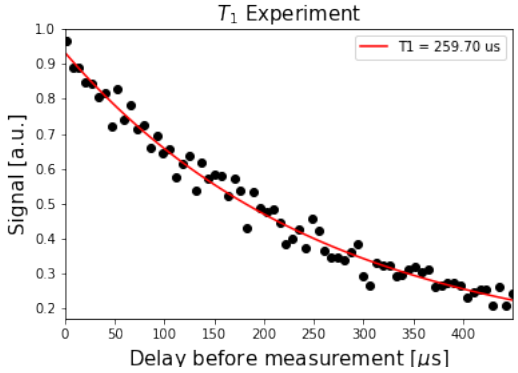


Fig 2.5: andamento della frazione di misure che hanno restituito |1> come risultato di ogni esperimento in funzione del tempo di ritardo inserito tra il gate X e la misura

Come si può notare dalla figura 2.5, la funzione esponenziale decrescente ben rappresenta la frazione di qubit che popola lo stato |1> in funzione del ritardo inserito; inoltre, il fit conferma quanto anticipato nel capitolo 1: i valori caratteristici di T1 per qubit superconduttori sono dell'ordine delle centinaia di microsecondi.
Per completezza, si annota che durante il lavoro di tesi è capitato di osservare oscillazioni di T1 tra i 150μs e i 260μs; ad indicare l'enorme delicatezza di questi sistemi. Segue che, sul lungo periodo, si sono osservate variazioni percentuali di circa il 27%.

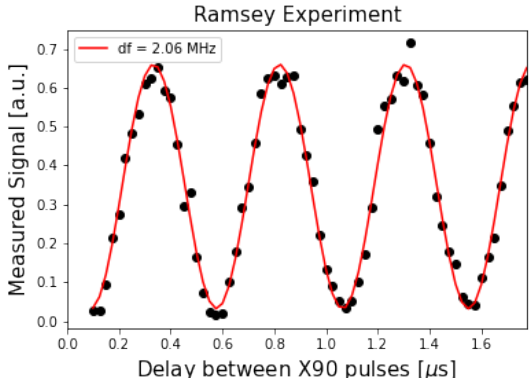
2.5 Ramsey Experiment

Come anticipato nel paragrafo "frequenza di risonanza", viene ora mostrato come stimare con maggior precisione tale frequenza. La procedura standard a tale fine prevede di modificare di qualche MHz la frequenza dell'impulso in ingresso al qubit (portandolo quindi fuori-risonanza, *detuning*) e con questa nuova frequenza inviare due impulsi π/2 intervallati da tempi di ritardo via via crescenti;

successivamente si 'chiude il circuito' con una misura. I risultati ottenuti mostrano un andamento sinusoidale in funzione della durata dei ritardi; questi dati vengono fittati con una funzione del tipo:

$$Acos(2\pi Bx - C) + D$$

dove: A è l'ampiezza della sinusoide, B (df in figura) è il parametro il cui valore indica di quanto il nostro segnale è fuori risonanza col qubit, C è una fase iniziale, D un termine costante.



Così facendo, si è ora in grado di dare una stima più precisa della frequenza di risonanza semplicemente calcolando:

$$f_{precisa} = f_{approx} + detuning - B$$

Dove f_{approx} è la frequenza di risonanza stimata precedentemente, *detuning* è il valore di discrepanza che si è aggiunto per mandare il segnale in ingresso fuori-risonanza, B è il parametro ottenuto dal fit. Di seguito si riportano i valori di f_{approx} e $f_{precisa}$:

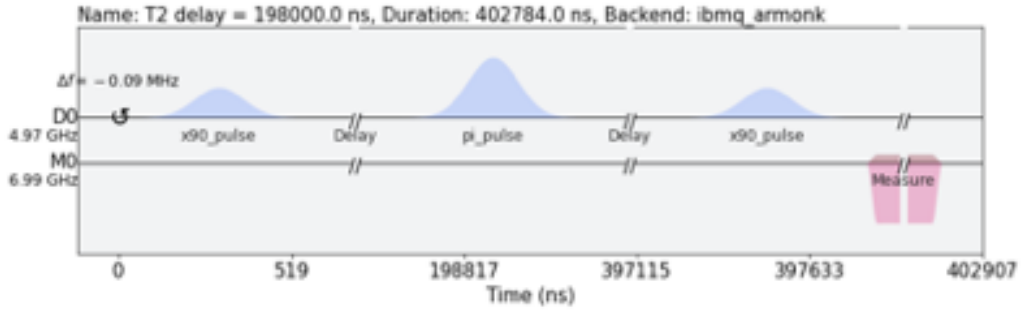
$$f_{approx} = 4.972\ GHz \quad f_{precisa} = 4.9714\ \text{GHz}$$

Come si può notare, l'esperimento Ramsey ha corretto la stima approssimata aggiungendo per fino una cifra significativa.

2.6 Tempo di coerenza (T2)

Il tempo di coerenza è definito come il tempo per il quale lo stato quantistico mantiene inalterata la differenza di fase relativa tra |0> e |1>; per questo, si parla anche di rilassamento trasverso, mentre il rilassamento responsabile del T1 è detto rilassamento longitudinale [1].
Per stimare il tempo di coerenza di un qubit si segue quello che in letteratura è detto "Hanh Echoes Experiment"; il quale si compone di una sequenza di impulsi π/2 - π - π/2 intervallati da un ritardo. Similmente all'esperimento Ramsey, si eseguono questi circuiti quantistici 500 volte per ogni ritardo e per 50 valori di ritardo diversi (da $4\mu s$ a $200\mu s$). Si vuol far notare che, preparato lo stato iniziale al livello fondamentale |0>, la sequenza di impulsi porterebbe di nuovo allo stato |0> in quanto, composti tra loro, equivalgono ad una rotazione di un angolo 2π attorno all'asse X. I ritardi, dunque, sono necessari per permettere solo stato quantistico di evolvere e creare così una differenza di fase

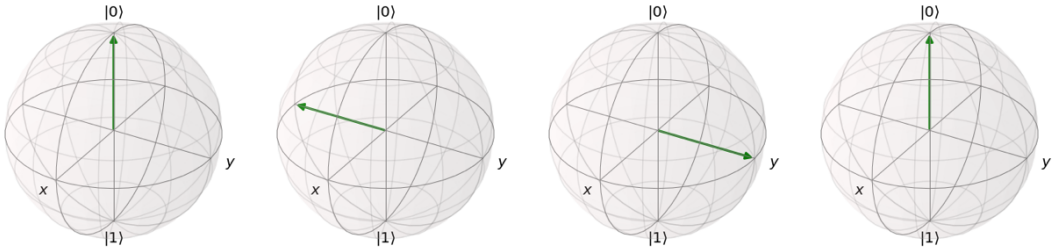
tra la sovrapposizione di stati |0> e |1>; possiamo visualizzare questo fenomeno come la precessione dello stato del qubit sulla sfera di Bloch.



Esempio:

Si mostrano ora due diversi scenari: il caso in cui i tre impulsi vengono inviati in sequenza immediata e il caso, durante il ritardo, si crea una differenza di fase relativa di π/2.

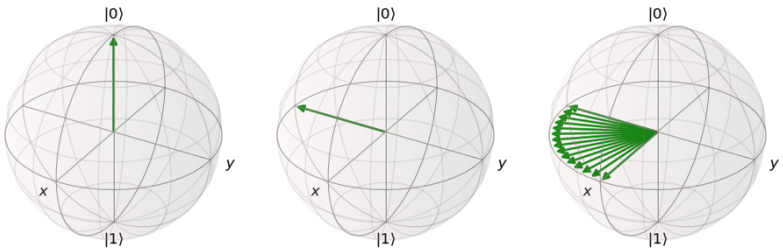
Caso 1: sequenza immediata



Impulso π/2 -> Impulso π -> Impulso π/2

La sequenza illustrata rende evidente quanto sostenuto in precedenza: applicare i tre gate senza alcun ritardo tra di essi riporta lo stato quantistico allo stato fondamentale. Tutte e tre le rotazioni vanno intese in senso antiorario attorno all'asse delle X.

Caso 2: precessione di π/2 durante il ritardo



Impulso π/2 -> Precessione durante il ritardo

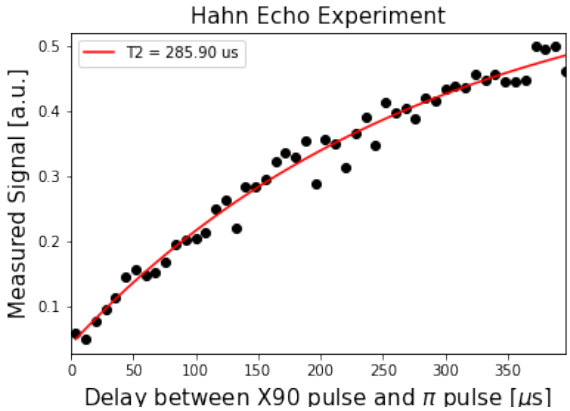
Si è mostrato ora un diverso scenario: dopo l'applicazione dell'impulso π/2 il vettore di stato ruota fino ad allinearsi con l'asse delle X, a questo punto una rotazione di angolo π attorno all'asse delle X sarebbe totalmente ininfluente. Bisogna osservare che questo secondo

scenario avverrebbe solo nel caso in cui fosse inserito un ritardo maggiore del T2; è un caso non contemplato nel nostro esperimento ma è stato scelto come esempio dell'effetto della precessione per motivi di chiarezza espositiva.

Le misure effettuate a termine di ogni singolo esperimento permettono di creare il grafico che mostra l'andamento della frazione di volte in cui si ottiene |1> in funzione dei tempi di ritardo inseriti. Ci si aspetta, per quanto detto prima, che la curva del grafico abbia origine all'intersezioni degli assi e poi cresca; questa crescita è di tipo esponenziale, dunque si fittano i dati con una funzione del tipo:

$$Ae^{-t/T2} + B$$

Dove T2 è il parametro di cui siamo interessati a conoscere il valore. Il valore ottenuto è la stima del tempo di coerenza.



Anche per questo parametro, durante il lavoro di tesi si sono ottenute stime molto diverse tra loro. Infatti, nel capitolo 1 è stato riportato il valore di T2 pari a 223,56μs; il giorno in cui si è scritto questo paragrafo il valore riportato nella scheda tecnica di ibmq-armonk riportava 274,57μs. Confrontando questi, con altri (tra cui quello mostrato in figura) si può affermare si aver osservato variazioni percentuali di circa il 25%.

Capitolo 3

Ottimizzazione dell'utilizzo di un qubit

Una volta appreso come manipolare un qubit (e.g. saper tradurre un gate logico in un preciso impulso), si deve pensare a come usarlo per il suo fine ultimo: essere utilizzato per il quantum computing. Questo ulteriore studio è di fondamentale importanza, in quanto, come dimostrato nel capitolo precedente, un qubit ha dei limiti intrinseci, tra i quali gli errori di implementazione di singolo gate e i tempi di decoerenza. L'utilizzo ideale di un qubit deve ridurre al minimo queste cause di errore.

3.1 Riduzione di errori dovuti ai gate

Per ridurre la probabilità che degli errori scaturiscano a causa dei gate inseriti in un algoritmo, ci sono due strade percorribili: lavorare a livello di hardware o a livello di algoritmo stesso; in questo paragrafo si discute questa seconda.
Come prima soluzione di ottimizzazione si può sfruttare un po' di semplice algebra lineare: il prodotto tra matrici 2x2. Questo perché ogni gate è una trasformazione unitaria applicata ad un sistema a due livelli, dunque rappresentabile con una matrice che gode, per definizione, della proprietà:

$$U^{\dagger}U = \mathbb{I}$$

Ragionando in termini "*gate=matrice*" si capisce come applicare più gate ad un qubit (vettore-stato di dimensione 2) equivalga a moltiplicare un vettore per un certo numero di matrici 2x2. Il risultato sarà comunque una singola trasformazione unitaria; infatti il prodotto di due matrici unitarie è una matrice unitaria. Inoltre, il prodotto tra matrici gode della proprietà associativa:

$$AB\ v\ =\ (AB)v$$

dove A e B sono matrici e v un vettore.
Analogamente si possono fare composizioni tra i gate (matrici unitarie) applicati ad un qubit (vettore-stato). Il risultato sarà un unico gate implementabile in un tempo minore rispetto a quello necessario per implementare tutti i gate originari: un gate U3 con opportuni parametri. Schematicamente:

$$G_n G_{n-1} \dots G_2 G_1 |\psi> \rightarrow\ U3(\theta, \varphi, \lambda)|\psi>$$

Dove G_n rappresenta il gate *n-esimo* applicato al qubit.
Per vedere come la riduzione di numero di gate comporti un vantaggio in termini di costo computazionale si riporta un esempio costruito utilizzando simulatori di computer quantistici.

Esempio:

Creo una generica sequenza (*seq1*) di gate i già noti X, Rz, P e Ry da applicare ad un qubit. Di seguito una rappresentazione (fig. 3.1).

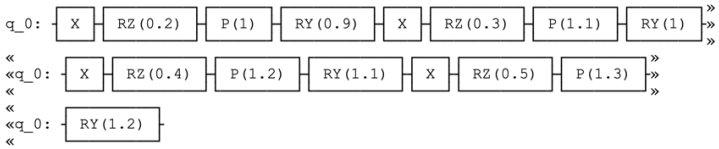
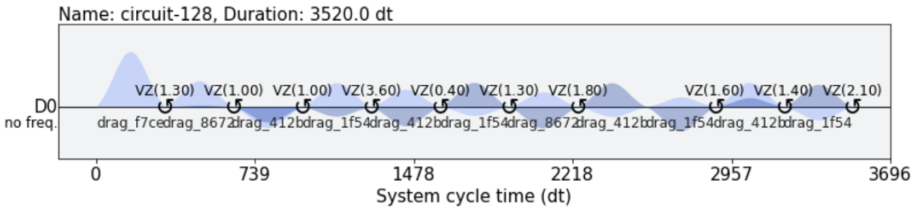
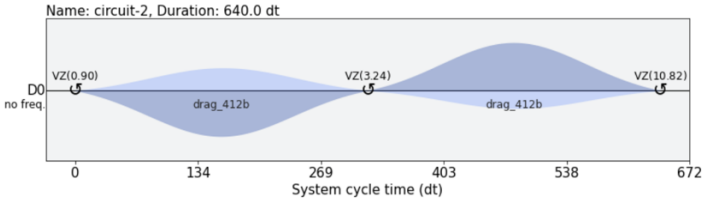


Fig. 3.1: Una sequenza arbitraria di gate applicati al qubit *q_0*

Utilizzando un simulatore (in questo caso *FakeArmonk*, in dotazione con le librerie Qiskit) mostro gli impulsi necessari per implementare questa sequenza:



Di questo grafico osservo un unico dato: la durata, in unità arbitrarie, pari a 3520dt.
Ora mostro gli impulsi necessari nel caso in cui la sequenza sia ridotta ad un unico U3 gate:



Ora la durata è di 640dt! La cosa interessante da osservare è che questa durata non dipende in alcun modo dalla lunghezza della sequenza originale e questo perché il gate U3 viene realizzato con al più tre rotazioni; che sono tre “operazioni” elementari che richiedono un unico impulso ciascuna.

Con l’esempio si è mostrato un caso in cui la riduzione di tempo di esecuzione è di circa un fattore 1/6; questo risultato, però, merita una precisazione. È stata ottenuta una riduzione notevole in virtù del fatto che la sequenza trasformata in singolo gate fosse particolarmente lunga. Di fatto, nella quasi totalità degli algoritmi vengono utilizzati più di un qubit e, in genere, non capita che si debbano applicare molti gate su un singolo qubit senza che questo venga fatto interagire con un altro; dunque, ci si può aspettare dei risultati più modesti da questa riduzione. Per chiarezza, si osservi la fig 3.2. rappresentante un circuito più “plausibile” di quello dell’esempio precedente.

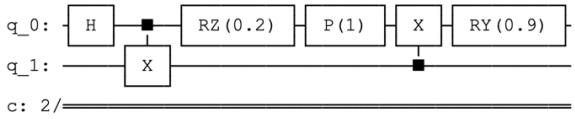


Fig 3.2: Nel circuito mostrato gli unici gate che possono essere composti sono Rz e P dato che i gate CNOT (indicati con una X e un segmento tra i due canali)implicano l’interazione tra qubit diversi.

Nelle librerie di Qiskit le composizioni tra gate su singolo qubit sono calcolate in automatico con il metodo *transpile().*

```
compiled_circuits = transpile(qc, backend=backend)
```

Questa semplice linea di codice restituisce un circuito compilato, in cui ogni sequenza di gate su singolo qubit è ridotta ad un unico U3 prendendo per argomento il circuito quantistico corrispondente ad un certo algoritmo e il backend sul quale si intende eseguire il circuito. Per completezza, è necessario specificare che il metodo *transpile()* faccia in realtà molto di più [19], in quanto adegua un algoritmo ad un determinato backend a seconda delle caratteristiche di ogni suo singolo qubit e della sua architettura interna; ma questo è un argomento che esula da quanto trattato nel capitolo.

3.2 Utilizzo di gate appena calibrati

Una volta minimizzato il numero di gate necessari per svolgere un determinato algoritmo quantistico, è bene massimizzare l'affidabilità di ogni singolo gate.
Per quest'ulteriore tecnica di ottimizzazione, non è necessario introdurre nulla di veramente nuovo: bisogna solo rifarsi al lavoro di calibrazione del capitolo 2. Si tratta, infatti, di sfruttare quanto fatto a livello hardware per avere vantaggi a livello di quantum computing. L'idea di fondo di questo capitolo è che i computer quantistici sono strumenti estremamente delicati e il loro utilizzo è fruttuoso solo a seguito di una rigorosa calibrazione [16]. Se non ci si volesse basare sulle calibrazioni giornaliere fatte da IBM (che, di conseguenza, hanno al più qualche ora di "età"), si può fare affidamento ad altre ancora più recenti: le nostre. Sì, perché quanto fatto su QiskitPulse può essere facilmente utilizzato in Qiskit. Ad esempio, la linea di codice necessaria per aggiungere un X gate al primo qubit di un *QuantumCircuit* chiamato qc è:

```
qc.x(0)
```

In questo modo non si hanno informazioni sull'impulso di microonde inviato al qubit, si utilizza quello implementato con l'ultima calibrazione fatta al computer. Per usare, invece, l'impulso coi parametri appena ottimizzati si devono usare i seguenti metodi:

```
with pulse.build(backend) as pi_pulse:
    drive_duration = get_closest_multiple_of_16(pulse.seconds_to_samples(drive_duration_sec))
    drive_sigma = pulse.seconds_to_samples(drive_sigma_sec)
    drive_chan = pulse.drive_channel(qubit)
    pulse.play(pulse.Gaussian(duration=drive_duration,
                              amp=pi_amp,
                              sigma=drive_sigma,
                              name='pi_pulse'), drive_chan)
qc.add_calibration('x', [0], pi_pulse)
```

In cui durata, sigma e ampiezza dell'impulso gaussiano sono i parametri ottenuti con le procedure spiegate nel capitolo 2. Così facendo, si controlla pienamente la corrispondenza gate←→impulso. Dalla chiamata del metodo *qc.add_calibration()* il programma smetterà di usare il gate pre-impostato ed inizierà a usare l'impulso creato dall'utente.

Per avere un confronto tra i diversi approcci è stato eseguito un esperimento. Sempre utilizzando il backend ibmq-armonk, si sono eseguiti dei circuiti elementari (applicazione di un X gate e misura del qubit) in tre modi differenti:

1. Utilizzando le librerie Qiskit. Si tratta semplicemente di dichiarare un oggetto della classe QuantumCircuit, applicare un X gate seguito poi da una misura. Il risultato atteso di questo circuito è, ovviamente, lo stato |1>, per stimare l'affidabilità di questo gate preimpostato si è commissionato al servizio cloud di IBM l'esecuzione di questo algoritmo 30 volte con 1000 tentativi ciascuno per ottenere sufficiente statistica.
2. Utilizzando le librerie QiskitPulse. In questo secondo caso si è ripercorso il programma di caratterizzazione fino alla dichiarazione del π-pulse; giunti a questo punto non si è proceduto con la stima di T1 e T2 ma si è costruito il medesimo circuito descritto nel punto 1; in questo, però, l'X gate è stato implementato con l'impulso appena calibrato. Il numero di esecuzioni richieste è pari a quelle precedenti.
3. Utilizzando le librerie QiskitPulse ma con un lavoro di calibrazione più completo. Come anticipato nel Capitolo 2, è possibile affinare la calibrazione del π-pulse iterando il processo. Di fatti, si è mostrato come costruire questo impulso utilizzando solo una stima approssimativa della frequenza di risonanza; questo perché si è immaginato di non esser ancora in grado di passare da |0> a |1>, il che sarebbe poi stato necessario per i vari esperimenti come il Rabi e il Ramsey. Ma, seguendo quanto descritto in [16], ai fini di una migliore ottimizzazione dei parametri di questo impulso è consigliato seguire i seguenti step:
   - Trovare una prima stima di frequenza di risonanza utilizzando tecniche di spettroscopia all'output del qubit; come fatto nel paragrafo "Frequenza di risonanza" del Capitolo 2.
   - Impulsare il qubit a questa frequenza in modo da osservare le oscillazioni Rabi e ottenere da un fit con una funzione sinusoidale l'ampiezza necessaria per implementare un π-pulse.
   - Eseguire un esperimento di Ramsey in modo da ottenere una stima più precisa della frequenza di risonanza.
   - Eseguire un ulteriore esperimento per osservare le oscillazioni Rabi ed estrarre da un nuovo fit l'ampiezza definitiva dell'impulso necessario per passare dallo stato |0> allo stato |1>.

   Anche per questo terzo metodo sono stati eseguiti 30 volte set di 1000 circuiti ripetuti.

In tabella si riportano i risultati dei tre esperimenti; in particolare si mostrano i valori medi del numero di volte in cui si è ottenuto |1> su un set di 1000 esecuzioni.

Metodo 1	Metodo 2	Metodo 3
904 ± 2	903 ± 2	904 ± 2

Si reputano questi risultati estremamente indicativi delle prestazioni del qubit e della bontà delle calibrazioni giornaliere di IBM. La prima cosa che si vuole osservare è che si ottiene |1> come risultato della misura solo nel 90% dei casi (circa). Questo è piuttosto sorprendente. Di fatti si è considerato il circuito più semplice da eseguire, in cui compare un unico gate che non ha nemmeno bisogno di essere scomposto in operazioni più elementari; ci si poteva aspettare un valore decisamente più prossimo a 100.
In seconda battuta, si vuole commentare la perfetta compatibilità dei tre metodi utilizzati. L'unico a sembrare un po' meno performante rispetto agli altri sembra essere quello che si basa sulla stima "grezza" della frequenza di risonanza. C'è però un dettaglio che si vorrebbe rimarcare: per svolgere

questa tesi non si ha avuto a completa disposizione un qubit, tutti i circuiti inviati al servizio cloud sono stati messi "in coda" con altri di altri utenti e questo va a discapito delle calibrazioni fatte con QiskitPulse e poi utilizzate nei circuiti. Il caso ideale, che si è descritto a inizio paragrafo, prevede che le esecuzioni dei circuiti siano immediatamente successive alle calibrazioni, il che potrebbe portare una maggiore affidabilità del metodo 3.
Altro aspetto non trascurabile per il commento dei risultati è l'errore nella misura dello stato di un qubit. Nella scheda tecnica del computer ibmq-armonk si legge *"Avg. Readout Error: 3.86e-2"* [13]. Il che significa che l'errore di lettura dello stato di un qubit è affetto da un'incertezza relativa nell'ordine del 4%! Il che è ben al disopra delle deviazioni standard dalla media riportate in tabella. Per poter dire in che misura i risultati ottenuti siano dovuti all'errata implementazione del gate o all'errore di lettura dello stato, bisognerebbe condurre ulteriori approfondimenti. Questi, però, esulano dallo scopo di questo lavoro interessato al confronto tra gate preimpostati e gate creati dall'utente; i quali sono stati qui confrontati a parità di hardware, dunque a parità di apparato condizioni.

3.3 As Late As Possible (ALAP)

"*As Late As Possible"* è l'approccio alla schedulazione utilizzato di default da Qiskit. Sebbene si possa utilizzare il suo opposto "*As Soon As Possible",* c'è un ottimo motivo fisico per scegliere l'approccio ALAP: la decoerenza. L'idea di base è quella che un qubit lasciato allo stato fondamentale ha poche probabilità di cambiare di stato, mentre un qubit in una sovrapposizione di stati o esattamente nello stato |1> è soggetto sia a rilassamento trasverso che longitudinale.
Chiaramente, questo discorso è valido esclusivamente per circuiti in cui è presente un numero plurale di qubit, in cui è possibile che solo un certo sottogruppo venga utilizzato nella prima parte di algoritmo e solo più tardi vengano "chiamati in causa" i restanti.

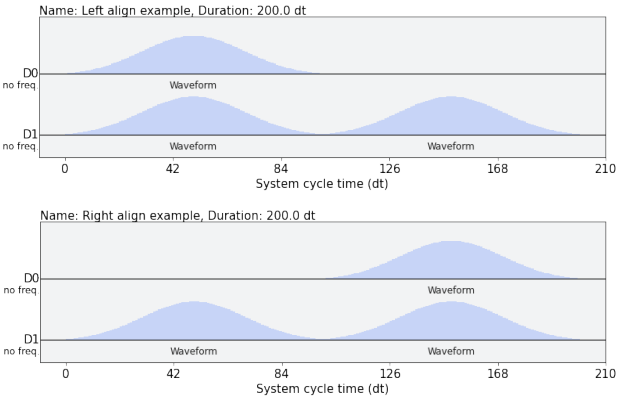


Fig. 3.2: In figura sono rappresentati i due diversi approcci di schedulazione ALAP e ASAP. Importante notare come nel caso destro (ALAP) nessun qubit venga lasciato interagire liberamente con l'ambiente dopo il primo impulso.

3.4 Riduzione della durata singolo gate

Un possibile modo di ottimizzazione dell'utilizzo di un qubit è la riduzione dei tempi di implementazione di singolo gate. Ridurre il tempo di esecuzione di un dato circuito quantistico, infatti, significa ridurre la probabilità che si verifichi decoerenza o *dephasing* dello stato del qubit.
Si mostrano in seguito i risultati ottenuti da un esperimento di stima dell'accuratezza dell'implementazione dell'X gate per diversi valori di durata di impulso. Con le librerie di Qiskit è infatti possibile utilizzare dei gate con i parametri d'impulso (durata, forma e ampiezza) calibrati dall'utente. Come trovare questi parametri è stato mostrato nel capitolo 2. L'esperimento dal quale si sono ottenuti i risultati prevede di scegliere a priori la durata e la forma dell'impulso e di ottimizzare il valore in ampiezza affinché l'impulso comporti una rotazione di π attorno all'asse delle X (ossia faccia passare da |0> a |1> e viceversa). La forma dell'impulso scelta è quella gaussiana e per le durate dell'impulso ne sono state scelte 3: quella standard delle librerie Qiskit per il backend utilizzato (8 sigma da 0,075μs), una ridotta del 60% (sigma = 0,030μs) e una dell'80% (sigma = 0,015μs). Ulteriori riduzioni non sono state possibili in quanto necessitavano ampiezze d'impulso superiori a quelle consentite dall'hardware.
Trovati i relativi valori d'ampiezza si è proceduto stimando l'accuratezza dei gate implementati svolgendo ripetutamente (30 volte jobs di 1000 shots) il circuito quantistico X gate e misura. Da questi dati si è stimato quante volte, in percentuale, il risultato del circuito fosse quello atteso, ossia lo stato |1>.

Valori ottenuti:

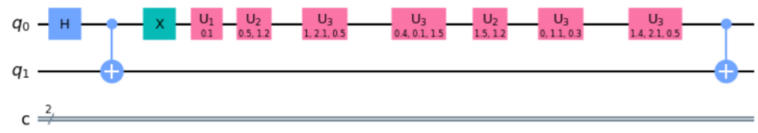
Durata sigma (μs)	Accuratezza percentuale
0,075	91,2 $\pm$ 0,2 %
0,030	91,0 $\pm$ 0,2 %
0,015	91,2 $\pm$ 0,4 %

Questi risultati sembrano essere in contrasto con quanto riportato dalla tesi [16] la quale afferma che per qubit superconduttori l'accuratezza dei gate decresce riducendo la durata dell'impulso che lo implementa.
A giustificare questi risultati si sono fatte diverse ipotesi:

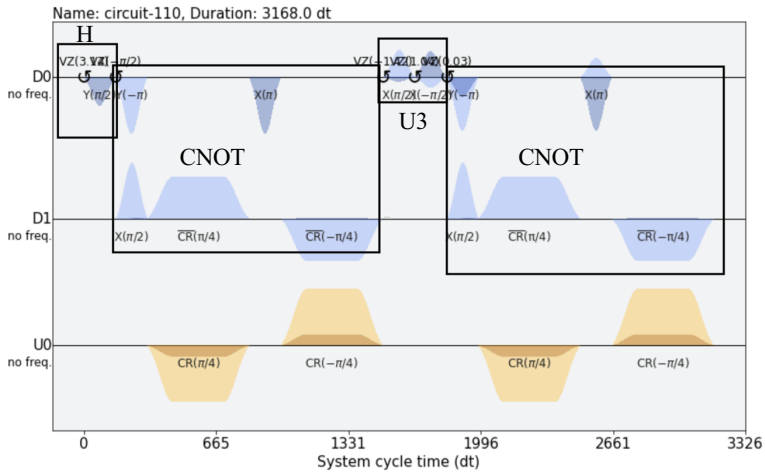
1. La durata del gate non è stata ridotta a sufficienza per poter apprezzare una diminuzione di accuratezza. Se così fosse, però, sarebbe da considerarsi comunque notevole esser stati in grado di ridurre dell'80% la durata di un gate senza perdere in accuratezza.
2. L'errore medio di misura dello stato (Avg. Readout error) domina le incertezze sui dati; nel caso di ibmq-armonk è pari a 3.86e-2. Di conseguenza i risultati ottenuti non sarebbero indicativi della bontà dell'implementazione del gate.
3. Nel quantum computing a più qubit non si è molto interessati a ridurre il tempo di implementazione di gate su singolo qubit in quanto poi le operazioni vanno svolte in parallelo e non sia quindi conveniente avere un qubit sul quale non sta agendo nessun impulso. Inoltre, la durata di gate quali CNOT e altri gate che coinvolgono più qubit hanno durata maggiore.

Proprio per indagare quest'ultimo aspetto si mostrano le schedulazioni d'impulso di alcuni circuiti che richiedono 2 qubit.

Esempio 1: applicazione di H gate, CNOT, una serie arbitraria di rotazioni sul primo qubit e ulteriore CNOT.

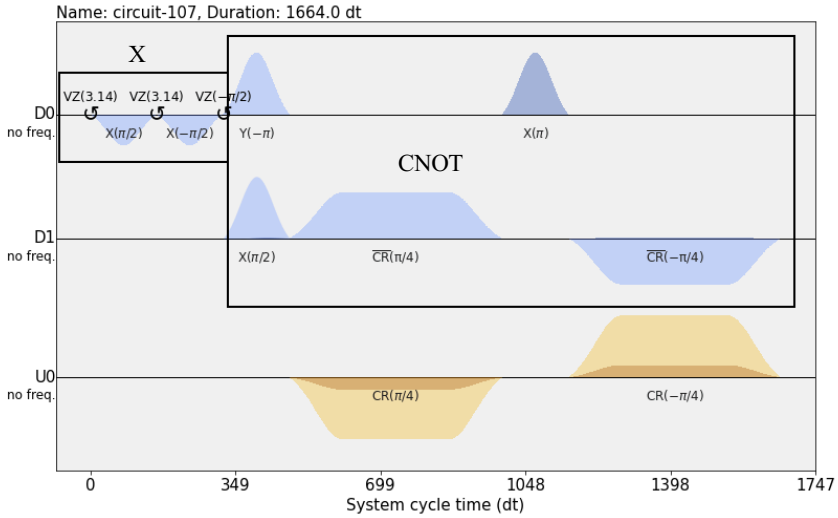


Circuito



Schedulazione d'impulsi: con riquadri neri si sono evidenziati gli impulsi che implementano un gate.

Esempio 2: Un circuito composto solo da X gate e un CNOT. Di questo si potrebbe ridurre la durata dell'X gate dell'80%.



A sostegno della terza ipotesi si mostra il confronto tra un X gate e un CNOT in schedulazione d'impulsi.

Dato che i risultati ottenuti sembrano smentire quanto atteso, si è deciso di eseguire un nuovo esperimento in cui ad un singolo qubit vengono applicati 11 X gate. Anche per questo circuiti il risultato atteso è lo stato |1> ma ci si aspetta che un errore di calibrazione di un gate apporti una maggiore discrepanza tra le stime di accuratezza. Importante sottolineare come in questo caso non sia

stato utilizzato il metodo transpile di Qiskit in quanto avrebbe ridotto l'intero circuito ad un singolo X gate. In questo esperimento sono stati raccolti dati per l'X gate preimpostato e per tre di diversa durata (sigma pari a 0,075μs - 0,025μs - 0,015μs) calibrati con la procedura già citata.
I risultati ottenuti sono riportati in tabella.

Durata sigma (μs)	Accuratezza percentuale
0,075 (preimpostato)	88,6 $\pm$ 0,2 %
0,075 (calibrato)	88,5 $\pm$ 0,2 %
0,030	88,5 $\pm$ 0,3 %
0,015	88,7 $\pm$ 0,2 %

Anche in questo caso tutti e quattro i valori di accuratezza stimati sono compatibili tra loro. Questo porterebbe a dire di esser stati in grado di ridurre notevolmente la durata d'implementazione di un X gate. Ulteriori considerazioni sono lasciate alle conclusioni finali.

Capitolo 4

Quantum Classifier

È stato dimostrato che un singolo qubit *ideale* può essere usato come classificatore universale [20]; scopo di questo capitolo è ripercorrere tale dimostrazione e discutere la difficoltà di implementare un tale classificatore su un device quantistico *reale.*

4.1 Introduzione al problema generale

I classificatori universali esistono già nell'informatica classica e vengono implementati con tecniche di machine learning. In generale, la classificazione è un problema che consiste nell'identificare a quale categoria appartiene un elemento in input, sulla base di un modello di classificazione ottenuto in apprendimento automatico. Tipicamente il modello di classificazione è ottenuto istruendo la macchina tramite "l'apprendimento supervisionato"; ossia, avvalendosi di un set di dati usato per il *training* contenente degli esempi classificati correttamente. Gli algoritmi di classificazione possono essere lineari o non lineari: nel primo caso la separazione tra le classi è una retta o un piano, nel secondo è una curva [21]. In questo capitolo verranno considerate classificazioni non lineari.
Per esempio, la classificazione di una linea chiusa è un problema non banale per una *rete neurale* che lavora in regime lineare; in questo caso, per raggiungere un livello di classificazione soddisfacente, è necessario avere a disposizione un gran numero di neuroni [20]. Ed è questo il motivo per cui può essere vantaggioso usare un qubit come classificatore. Infatti, i *layer* di un classificatore a singolo qubit sono i quantum gate che, come discusso nel capitolo 1, possono essere intesi come rotazioni, ergo: operazioni intrinsecamente non lineari.
La tecnica proposta per implementare un quantum classifier è il "*Data re-uploading"*, un'idea che sconvolge un po' quella del solito circuito quantistico basato sullo schema "inizializzazione dello stato – quantum gates – misure". Il data re-uploading prevede infatti di caricare dei dati ripetutamente *durante* l'esecuzione del circuito. Questo è stato pensato in analogia al funzionamento di una classica rete neurale. In questa, infatti, ogni neurone riceve in input dati da tutti i neuroni precedenti; procedura che nel mondo quantistico non è consentita a causa del *non-cloning theorem* [2] . Per aggirare questa limitazione è possibile ricaricare i dati classici nel circuito quantistico più volte in modo tale che ogni layer li riceva. In figura (fig. 4.1) viene mostrato uno schema esemplificativo.

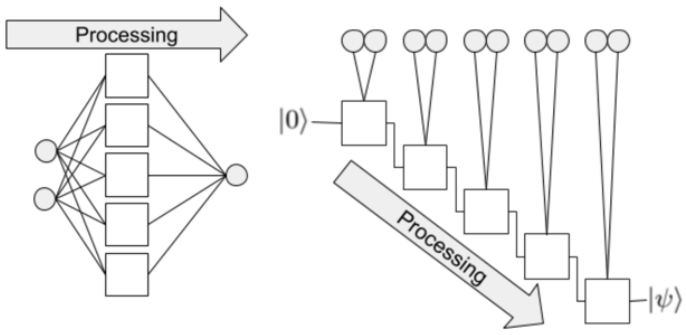
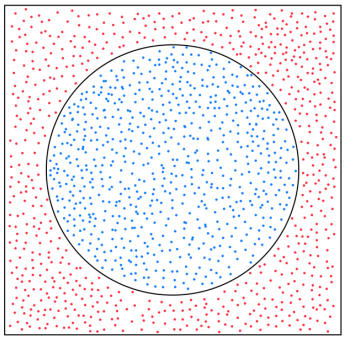


Fig. 4.1: A sinistra lo schema seguito dalle reti neurali, a destra quello del data re-uploading. Si evidenzia come in entrambi gli schemi ogni layer (quadrati) riceva in input i dati (palline).

4.2 Problema affrontato

Per testare l'efficacia del quantum classifier si inizia con un esempio: la classificazione di un cerchio. Il set di dati usato per il training consiste in 400 punti $(x_1\ ,\ x_2)$ presi random in un piano; ognuno etichettato come 0 (blu) o 1 (rosso) a seconda che si trovi all'interno o all'esterno di un cerchio di raggio per cui la sua area risulti essere pari a metà superficie del piano.



L'obbiettivo è di essere in grado di predire se un punto si trova all'interno o all'esterno della circonferenza dati in input le sue coordinate.
Per tradurre questo problema in un circuito quantistico si ricorda che lo stato di un singolo qubit è un vettore bidimensionale ad entrate complesse (con la dovuta normalizzazione) e può essere interpretato come un punto sulla sfera di Bloch. Per caricare i dati classici (e.g. le coordinate di un punto) in un qubit si usano trasformazioni unitarie del tipo $U(x_1, x_2, x_3)$; applicate allo stato iniziale |0>. Dato che si sta affrontando un problema bidimensionale le operazioni unitarie di questo esempio saranno del tipo $U(x_1, x_2, 0)$ e comportano uno "spostamento" da un punto ad un altro sulla sfera di Bloch.

4.3 Modellizzazione del problema

Una volta caricati i dati all'interno del circuito quantistico, si creano dei modelli non lineari su cui fare training simili ai "pesi" delle reti neurali. Nel caso qui analizzato sono ampiezze di angoli di rotazione attorno agli assi Y e Z (sempre in riferimento alla sfera di Bloch). Queste operazioni unitarie susseguono al caricamento dei dati e la composizione di queste due operazioni restituiscono un'unica unitaria (come discusso nel capitolo 2). Chiamiamo layer questa unica operazione composta, la cui forma è data dal prodotto:

$$L(\vec{\theta}\ ,\vec{x}) = U(\vec{\theta}\ )U(\vec{x}\ )$$

Al fine di aumentare il numero di parametri ottimizzabili (come analogamente si fa aumentando il numero di neuroni in una rete neurale), è possibile riapplicare questi layer più volte con dei nuovi set di pesi $L(\vec{\theta}_L\ ,\vec{x})\ L(\vec{\theta}_{L-1}\ ,\vec{x})\ldots\ L(\vec{\theta}_2\ ,\vec{x})\ L(\vec{\theta}_1\ ,\vec{x})$ per un totale di L layer. Lo stato finale del circuito quantistico sarà dunque dato da:

$$|\psi>= L(\vec{\theta}_L\ ,\vec{x})\ L(\vec{\theta}_{L-1}\ ,\vec{x})\ \ldots\ L(\vec{\theta}_2\ ,\vec{x})\ L(\vec{\theta}_1\ ,\vec{x})|0>$$

Le operazioni fin qui descritte sono prettamente lineari; mentre, per implementare un classificatore universale è necessario introdurre termini non lineari [22]. Con l'applicazione di tutti i layer lo stato

ottenuto |ψ> è identificabile con un punto sulla sfera di Bloch e questo deve in qualche modo essere classificato come 0 o 1 (interno o esterno al cerchio). Non avendo diretto accesso allo stato è necessario applicare una misura al qubit la quale restituirà |0> o |1>; questo è il passaggio non lineare. Data la natura statistica di quest'operazione, è necessario svolgere l'intero processo un gran numero di volte prima di potere stimare la *fedeltà* (o, in un certo senso, vicinanza) dello stato creato con quello che rappresenta la classe di appartenenza. Gli stati ottenuti con diversi pesi (i parametri θ dei layer) daranno valori di fedeltà diversi e di questi verranno selezionati quelli che minimizzano la vicinanza allo stato-obbiettivo.
Al fine di ottenere stati il più vicini ai rappresentanti delle classi (in questo caso |0> o |1>) si utilizzano tecniche di minimizzazione tipiche del machine learning. Viene dichiarata una *cost function* della forma:

$$Cost = \sum_{Punti\ creati} (1 - < \psi | Obbiettivo >)$$

Dalla minimizzazione di questa si ottengono i parametri del modello di classificazione. Nella sommatoria è stato utilizzato il prodotto scalare < | > come se si avesse a disposizione l'esatta conoscenza dello stato |ψ>; ma quello che si intende è il calcolo di fedeltà spiegato in precedenza. La scelta di questa notazione sarà più chiara in seguito quando verrà presentato il caso di simulatore *ideale*.

4.4 Quantum classifier di Qibo

Ora si discute la possibilità di implementare effettivamente un circuito quantistico basato sul data re-uploading al fine di creare un classificatore universale. Per questo lavoro ci si è basati sui codici esempio di Qibo [23] [24], una libreria di Python per simulazioni di algoritmi quantistici. In particolare, si fa riferimento ai codici dell'esempio "*reuploading_classifier*" ispirato allo stesso articolo [20]. Questo fornisce un algoritmo variazionale per classificare dati classici usando un solo qubit e questo fine viene raggiunto con la tecnica del data re-uploading discussa prima. I gate scelti per essere ottimizzati al fine della classificazione sono del tipo $R_z(wx_1 + b)$ e $R_y(vx_0 + a)$, sebbene questo non sia l'unico ansatz considerabile. Con tecniche di minimizzazione come *sgd* (stochastic gradient descent) o L-BFGS-B (Limited-memory Broyden-Fletcher-Goldfarb-Shanno Boxed algorithm) si cercano poi i parametri migliori per massimizzare la fedeltà di uno stato rispetto allo stato da classificare. Il numero di parametri da ottimizzare può essere scelto dall'utente al momento della dichiarazione del numero di layer che intende utilizzare (da 1 a 10); di norma, un numero maggiore di layer porta ad una maggiore accuratezza della classificazione.
L'esempio di Qibo fa una simulazione *ideale* del circuito quantistico; il quale trasforma, con una serie di gate e *reupload* dei dati, lo stato iniziale in uno stato |ψ> conosciuto esattamente; ossia, la simulazione restituisce i coefficienti complessi α e β del vettore di dimensione 2 rappresentante lo stato. In questo modo, il calcolo della vicinanza con lo stato che definisce classe è calcolabile come vero prodotto scalare in notazione di Dirac $< \psi_1 | \psi_2 >$. Il cui risultato, in modulo, varia da 0 a 1 con 0 in caso di ortogonalità tra stati e 1 in caso questi coincidano.
Assumendo la conoscenza dei coefficienti α e β, l'esempio *reuploading_classifier* riesce a classificare diverse forme corrispondenti a un numero diverso di classi. La classe più semplice è chiamata *circle* e i due stati rappresentanti sono esattamente |0> e |1>; per problemi a maggior numero

di classi i vari stati-bersaglio sono stati scelti in modo che siano il più ortogonali possibile tra loro (fig. 4.3).

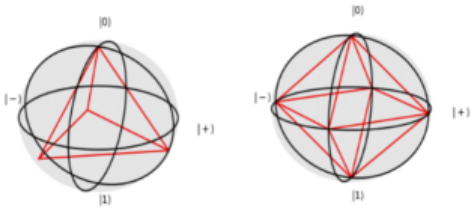


Fig. 4.3: Per problemi a più classi gli stati-obbiettivo sono i vertici di poliedri regolari.

I risultati di questo algoritmo sono forniti in termini percentuali (previsioni corrette/punti da classificare) e in modo grafico come in fig. 4.4.

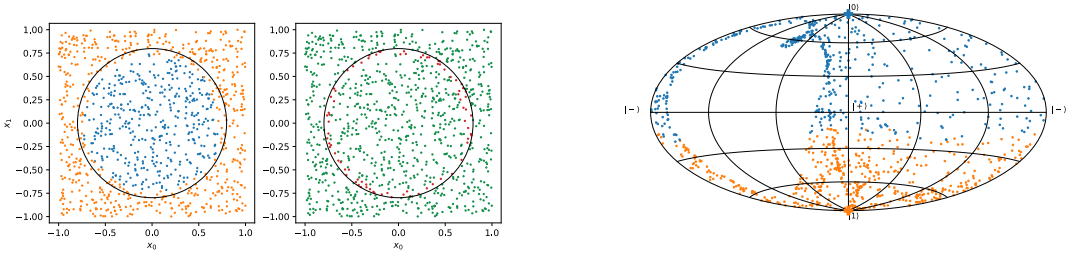


Fig 4.4a: Risultati grafici della classificazione dicotomica di |0> e |1>. Nel primo riquadro destra la definizione di classe (cerchio) in cui si vedono i punti che il modello allenato ha previsto essere interni (blu) e esterni (arancione); nel secondo riquadro si mostrano le previsioni corrette (verde) e quelle errate (rosso). A sinistra i punti sulla sfera di Bloch. Questa simulazione è stata fatta con 10 layer e ha riportato un'accuratezza del 91,2%.

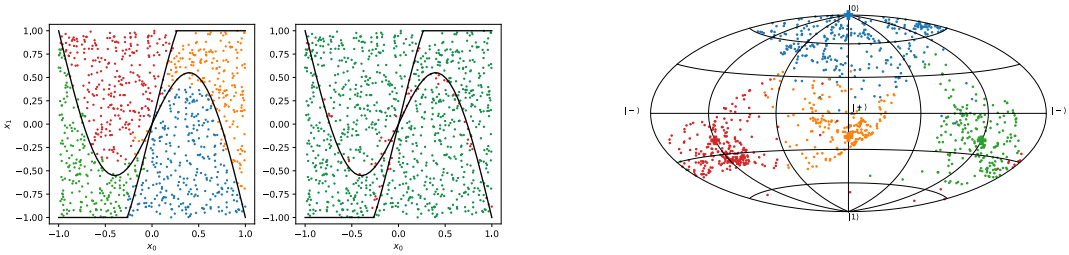


Fig. 4.4b: Qui rappresentata la classificazioni delle classi *wavy lines* dalla quale è stata ottenuta una percentuale di previsioni esatte dell'88,9%.

4.5 Quantum classifier in Qiskit

I codici dell'esempio di Qibo sono organizzati in tre file diversi:

1. datasets.py contiene le funzioni necessarie per creare il set di dati e le relative rappresentazioni
2. qlassifier.py codifica i circuiti quantistici e le loro esecuzioni
3. main.py è il file da cui attivare la classificazione dopo aver scelto il numero di classi e di layer.

Per questa tesi, il cui lavoro si basa sulle librerie di Qiskit e QiskitPulse, è stato necessario riscrivere la classe *single_qubit_classifier* contenuta nel file qlassifier.py. Si sono dunque sostituiti i metodi di Qibo con quelli di Qiskit. Il primo approccio è stato quello di ricreare una classe identica, che utilizzasse una simulazione ideale dalla quale ottenere il vettore esatto in output dal circuito quantistico; successivamente ci si è posti il problema di adattare questo esempio ad un vero backend quantistico. Sia una che l'altra riscrittura delle classi sono pubblicate nel repository Github [25].
La prima riscrittura ha richiesto l'utilizzo della libreria Statevector di Qiskit; nella quale lo stato del qubit è un vettore di dimensione due a coefficienti complessi e i gate sono matrici unitarie. Questa ha riportato risultati del tutto analoghi a quelli già ottenuti con l'esempio di Qibo.
Come primo passo verso l'implementazione di un classificatore quantistico si è scelto di utilizzare i simulatori della classe Aer di Qiskit; in particolare, i circuiti sono stati eseguiti sul backend *aer_simulator*, il più adatto nel caso si devono simulare circuiti che terminano con un'operazione di misura sui qubit. Su questo si è testato l'algoritmo per il task più semplice, ovvero la classificazione di un cerchio. Per fare questo, si è ulteriormente modificato il file qlassifier.py, prima basato sulla classe Statevector, in modo che i circuiti terminassero con una misura. Questo significa abbandonare l'informazione completa dello stato quantistico (i coefficienti complessi α e β) e avere a disposizione solo la probabilità di ottenere |0> o |1>. Non avendo modo di conoscere la differenza di fase della sovrapposizione di |0> e |1> si è ulteriormente semplificato il problema: si è assunto che i coefficienti fossero reali; l'interpretazione grafica di quest'assunzione è che non si stanno più considerando punti distribuiti sull'intera sfera di Bloch ma solo su un meridiano. I coefficienti reali sono stati dunque stimati dalle ampiezze di probabilità restituite dalle misure fatte a termine dello stesso circuito ripetuto 1'000. Osservando, però, problemi di minimizzazione da parte del metodo L-BFGS-B, si è indagato il motivo per qui questo approccio non funzionasse. Il punto cruciale è che si stanno simulando dei circuiti eseguiti su backend *reali*. Nella tabella che segue vengono comparati i coefficienti reali degli stati finali stimabili con i metodi delle librerie di Statevector con quelli che si ottengono da simulazioni da parte di Aer a seguito di circuiti ripetuti 1'000 e 1'000'000 di volte.

Statevector		Aer 1'000 shots		Aer 1'000'000 shots	
α	β	α	β	α	β
0.88534197	0.46494041	0.88825672	0.45934736	0.88524233	0.46513008
"	"	0.88487287	0.46583259	0.88580302	0.46406141
"	"	0.87349871	0.48682646	0.88545356	0.46472788
"	"	0.88881944	0.45825757	0.88533779	0.46494838

In tabella si mostra un unico risultato delle librerie Statevector in quanto non varia tra un'esecuzione e un'altra. A commento di quelli ottenuti col simulatore Aer, invece, si vuol far notare che ripetere un circuito 1'000 volte può portare variazioni dell'ordine dell'1% alle ampiezze di probabilità, variazioni che decrescono all'ordine dello 0,01% per 1'000'000 esecuzioni. Dato che questi sono i dati in input alla funzione *fidelity* e che quelli ottenuti con un simulatore possono essere asintoticamente identici a quelli del caso ideale (Statevector) si può affermare che l'algoritmo proposto può funzionare con un numero sufficientemente elevato di *shots*. Non è stato però possibile testare un programma di classificazione con più di 10'000 shots in quanto necessiterebbe di una potenza di calcolo superiore a quella di un normale computer (per essere eseguito in tempi brevi, si intende). Con 10'000 chiamate, invece, si sono ottenuti risultati ben lontani da quelli ideali: accuratezza di predizione tra il 55% e il 57%; da considerare insoddisfacenti, in quanto, classificando in maniera casuale i punti si avrebbe un valore di aspettazione del 50%.

Si lasciano alle conclusioni ulteriori commenti a riguardo; qui ci si limita a mostrare i risultati ottenuti al fine di dare un confronto con quelli dell'esempio originale (fig. 4.5).

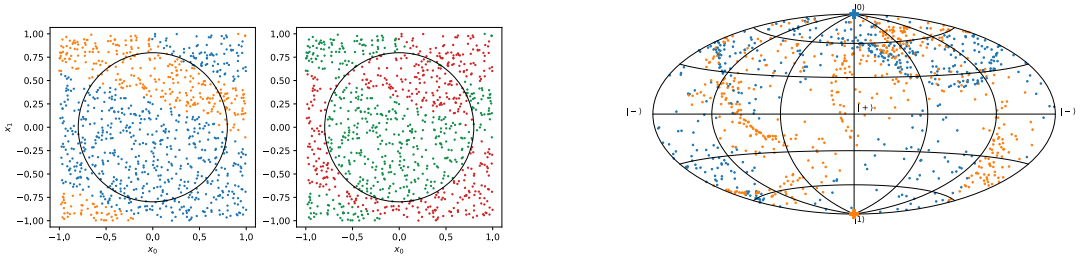


Fig 4.5: Con il backend Aer_simulator e usando 6 layer è si è raggiunta un'accuratezza della classificazione dell'56,2%. Il problema di minimizzazione ha richiesto più di un'ora.

Conclusioni

Con questo lavoro di tesi si è voluto offrire una panoramica dello stato dell'arte del quantum computing per quanto riguarda la sua unità fondamentale: il qubit. Sebbene gli aspetti teorici discussi nel capitolo 1 siano già consolidati, gli esperimenti mostrati nei capitoli seguenti, invece, mostrano quanta strada ancora c'è da fare per arrivare a sfruttare al meglio tutte le possibilità offerte da un singolo qubit.
Il lavoro di ottimizzazione proposto nel capitolo 3 sembra poter offrire una possibilità per migliorare le prestazioni di un computer quantistico. In particolare, sarebbe interessante riuscire ad estendere l'aspetto di calibrazione e di riduzione dei tempi di esecuzione a tutti i gate di un set universale. Infatti, in prospettiva di algoritmi di quantum computing, esser riusciti a ridurre dell'80% la durata dell'impulso relativo all'X gate, generalmente, non comporta significativi guadagni in termini di tempo di esecuzione di un intero algoritmo; in quanto gli X gate rappresentano una piccola frazione delle operazioni applicate ai qubit e sono i gate che coinvolgono più bit quantistici quelli che attualmente richiedono più tempo e portano errori maggiori.
Nel capitolo 4, invece, si è mostrato un algoritmo ideato e già dimostrato funzionante che ancora non può essere perfettamente eseguito da un device reale. Nonostante questo sembri essere uno dei *leitmotiv* del quantum computing attuale (quanto proposto in teoria non è ancora realizzabile in pratica), si è ritenuto interessante mostrare l'originale approccio del data re-uploading come esempio di buon utilizzo dei vantaggi offerti dalla meccanica quantistica in questo settore. La mancata esecuzione dell'algoritmo sul backend ibmq-armonk è dovuta a diversi motivi tra i quali: l'impossibilità di eseguire un circuito quantistico più di 2048 volte (shots), l'enorme numero di richieste di esecuzione al cloud che quest'esperimento avrebbe richiesto (probabilmente sarebbero serviti mesi per avere i risultati di tutte) e i risultati poco promettenti delle simulazioni.
Per quanto si sia mostrato ancora profondo il divario tra qubit ideale e qubit reale, si ritiene che il quantum computing sarà grande protagonista della fisica (e non solo) dei prossimi anni. Attualmente, circuiti quantistici vengono utilizzati per stimare l'energia dello stato fondamentale di molecole con l'algoritmo VQE (*Variational Quantum Eigensolver*) e si è già riusciti a scambiare delle chiavi crittografate in modo sicuro secondo il protocollo BB84 [27]. In futuro, con gli sviluppi del *Quantum Imaging*, si sarà in grado di effettuare radiografie meno invasive grazie all'utilizzo di radiazione meno energetica e dell'entanglement [28]. Il quantum computing, dunque, è un approccio diverso che non solo potrà essere vantaggioso ma, talvolta, necessario. Citando Feynman: *"Nature is quantum, goddamn it! So if we want to simulate it, we need a quantum computer."*

Appendice

In quest'appendice si approfondiscono aspetti teorici e si riportano tutti i programmi scritti e discussi nei vari capitoli omessi in precedenza per non appesantire oltremodo la lettura. L'ordine con cui si mostrano i vari argomenti rispecchia quello dei rimandi fatti nel testo.

A.1 Quantizzazione di una giunzione Josephson e Hamiltoniana di uno SQUID

Si riprende il discorso con la discussione delle giunzioni Josephson [11]. Una giunzione Josephson consiste in due superconduttori connessi con una barriera isolante per effetto tunnel. Può essere descritta tramite la sua corrente critica I_c e la differenza di fase *gauge* invariante φ attraverso la giunzione. Valori che caratterizzano la giunzione e dipendono dai materiali superconduttori impiegati e dalle dimensioni della stessa. Più precisamente, si può associare ad ogni superconduttore k=1,2 una funzione d'onda del tipo:

$$\Psi_k = \sqrt{\rho_k} e^{i\phi_k}$$

dove ρ_k è la densità di coppie di Cooper del k-esimo conduttore e ϕ_k la rispettiva fase. La dinamica del sistema è descritta dalle equazioni di Schrödinger:

$$i\hbar \frac{\partial \Psi_1}{\partial t} = E_1 \Psi_1 + k\Psi_2$$

$$i\hbar \frac{\partial \Psi_2}{\partial t} = E_2 \Psi_2 + k\Psi_1$$

dove E_1 e E_2 sono le energie degli stati e k la costante d'accoppiamento che misura l'interazione delle due funzioni d'onda. Sostituendo l'espressione di Ψ_k nelle equazioni di Schrödinger si ottengono le seguenti:

$$\hbar \frac{\partial \rho_1}{\partial t} = 2k\sqrt{\rho_1 \rho_1} sin\phi$$

$$\hbar \frac{\partial \rho_2}{\partial t} = -2k\sqrt{\rho_1 \rho_1} sin\phi$$

$$\hbar \frac{\partial \varphi}{\partial t} = E_2 - E_1$$

Le derivate $\frac{\partial \rho_1}{\partial t} = -\frac{\partial \rho_2}{\partial t}$ sono proporzionali alla *corrente di Josephson* I_J, mentre la quantità $2k\sqrt{\rho_1 \rho_1}$ alla corrente critica I_c menzionata precedentemente. Inoltre, se si applica una differenza di potenziale V alla giunzione, si ottiene $E_2 - E_1 = 2eV$ e la precedente equazione può essere riscritta nelle *equazioni di Josephson:*

$$I_J(t) = I_c sin\phi(t) \qquad \text{(prima eq. di Josephson)}$$

$$\frac{\partial\varphi(t)}{\partial t} = \frac{2\pi}{\Phi_0}V \qquad \text{(seconda eq. di Josephson)}$$

Con queste si è in grado di descrivere l'evoluzione temporale della corrente di Josephson e di φ in funzione dellla differenza di potenziale V applicata. Nell'ultima equazione scritta è stato introdotto il fattore $\Phi_0 = h/2e$ dove *2e* è la carica di una coppia di Cooper.
Derivando la prima equazione di Josephson rispetto al tempo e ricordando $\dot{I} = V/L$ si ottiene un'induttanza non lineare:

$$L_J = \frac{1}{cos\varphi}\frac{\Phi_0}{2\pi I_C}$$

L'energia relativa a questo termine d'induttanza la si può calcolare come:

$E_{J,L} = \int_0^t d\tau\, I_J(\tau)V = E_J(1 - cos\varphi)$; con $E_J = \frac{\Phi_0 I_C}{2\pi}$.

Dove E_J è chiamata energia di Josephson ed è una misura dell'accoppiamento attraverso la giunzione. Ma, dal momento che la giunzione di Josephson ha anche una capacità interna, indicata C_J, va considerato anche il relativo termine di energia: $E_{J,C} = \frac{Q^2}{2C_J}$ dove Q è la carica della giunzione.

A meno di termini costanti, l'Hamiltoniana classica è dunque:

$$H_J = \frac{Q^2}{2C_J} - E_J cos\varphi$$

Dato che Q=(2e)N, dove N è un intero che definisce il numero di coppie di Cooper presenti in eccesso, $N = N_1 - N_2$, con N_1 e N_2 a rappresentare il numero di coppie presenti su ogni faccia della giunzione, si può definire il termine energetico capacitivo $E_C = \frac{e^2}{2C_J}$ e riscrivere l'Hamiltoniana come:

$$H_J = 4E_C N^2 - E_J cos\varphi$$

Se invece di una singola giunzione Josephson se ne considerano due in parallelo, così da formare un anello superconduttore (o *loop*) si va a costituire un sistema chiamato SQUID: *Superconducting Quantum Interference Device*. Fig A.1

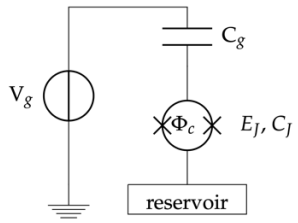


Fig A.1: Rappresentazione di uno SQUID inserito in un circuito con una differenza di potenziale V_g.

Nel caso in cui l'induttanza del loop può essere trascurata, allora l'Hamiltoniana corrispondente è identica ad H_J se ridefiniamo:

$$C_J \to 2C_J, \qquad E_J \to E_J(\Phi_C) = 2E_J cos\left(\pi \frac{\Phi_C}{\Phi_0}\right)$$

Il termine Φ_C è (l'eventuale) flusso esterno, il quale può alterare il valore di E_J.
D'ora in avanti consideriamo uno SQUID inserito in un circuito proprio come in figura. La presenza di V_g riduce N di $N_g = C_g V_g/(2e)$, così l'Hamiltoniana diventa:

$$H = 4E_C(N - N_g)^2 - E_J cos\varphi, \text{ con } E_c = \frac{e^2}{2(C_J + C_g)}$$

Considerando ora $4E_c N^2$ come termine cinetico e $-E_J cos\varphi$ come energia potenziale, allora H rappresenta l'Hamiltoniana di un oscillatore non lineare, in cui le variabili coniugate sono N (il momento corrispondente) e φ (l'analogo della posizione).
Per passare all'Hamiltoniana quantistica si procede sostituendo N e φ con i corrispondenti operatori. Dato che $\widehat{N}$ e $\hat{\varphi}$ sono variabili coniugate, nella base di autostati di $\hat{\varphi}$ vale la seguente relazione:

$$\hat{\varphi} \to \varphi \qquad \widehat{N} \to -i\frac{\partial}{\partial \varphi}$$

In questo modo si può riscrivere l'Hamiltoniana:

$$H = 4E_c\left(-i\frac{\partial}{\partial \varphi} - N_g\right)^2 - E_J cos\varphi$$

Le cui soluzioni al problema agli autovalori sono date dalle soluzioni di Floquet:

$$\psi_m(\varphi) = \frac{1}{\sqrt{2}} me_{-2[N_g - f(m,N_g)]}\left(-\frac{E_J}{2E_c}\frac{\varphi}{2}\right)$$

$$f(m, N_g) = \sum_{k=\pm 1} [int(2\,N_g + k/2) mod 2]\{int[N_g] - k(-1)^m[(m+1)div 2 + m mod 2]\}$$

dove int(x) arrotonda all'intero più vicino x, x mod y denota la solita operazione di modulo, ossia restituisce il quoziente intero di x/y. I corrispondenti autovalori sono:

$$E_m = E_c a_{-2[N_g - f(m,N_g)]}\left(-\frac{E_J}{2E_c}\right)$$

Dove $a_v(q)$ denota il valore caratteristico di Mathieu. In fig A.2 è riportato l'andamento di E_m (per m=0,1,2,3) in funzione di N_g e normalizzati rispetto all'energia di transizione E_{01}, la quale è la minima separazione energetica tra i livelli E_1 e E_0, per diversi valori di E_J/E_c.

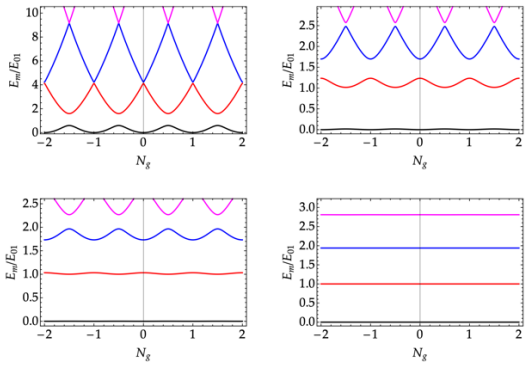


Fig A.2: E_m in funzione di N_g (in ogni grafico, dal basso verso l'alto m=0,1,2,3) normalizzati rispetto a E_{01} per diversi valori di E_J/E_c=1.0, 5.0, 10.0, 50.0. Il punto zero dell'energia è scelto come il minimo del livello m=0.

Si possono ora individuare due regimi: per $E_c \gg E_J$ si è in regime *charge* e per $E_c \ll E_J$ in regime *transmon.* Sebbene entrambi siano validi per la costruzione di un qubit, si discuterà solamente del regime transmon; in quanto questo è quello usato da IBM. I due regimi sono mostrati in fig A.3.

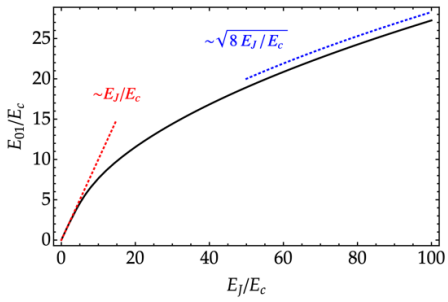


Fig A.3: grafico di E_{01}/E_c in funzione di E_J/E_c

4.2 Qubit di tipo transmon

Come mostrato in fig A.2, i livelli energetici E_m sono rappresentabili con una funzione oscillante che dipende da N_g. Ma, nel limite $E_J \gg E_c$ il termine oscillante diventa trascurabile e si può considerare che E_m non dipenda più da N_g. Questo è il regime *transmon*: *transmission line shunted plasma oscillation.*

Quando lo SQUID è portato a questo regime, il termine energetico capacitivo può essere considerato pari a:

$$E_c = \frac{e^2}{2(C_J + C_g + C_B)}$$

Per cui è possibile diminuire E_c aumentando C_B, fino ad ottenere $E_J \gg E_c$. A questo punto si può espandere in serie di Taylor $cos\varphi$ che compare nell'Hamiltoniana per ottenerne una "nuova" valida per i qubit di tipo transmon. Si fa notare che in questa non compare N_g in quanto i livelli energetici sono indipendenti da esso.

$$H_{Tr} = 4E_c\widehat{N}^2 + \frac{1}{2}E_J\hat{\varphi}^2 - \frac{1}{24}E_J\hat{\varphi}^4$$

In questa Hamiltoniana è facile individuare un oscillatore armonico tra i primi due termini e il temine non-lineare perturbativo $H_1 = -\frac{1}{24}E_J\hat{\varphi}^4$. Si dimostra come questo termine, in teoria delle perturbazioni, porti a non avere più livelli energetici equidistanti e si raggiungono le espressioni valide per i primi due salti di energia:

$$\Delta E_{0,1} = \sqrt{8E_JE_c} - E_c$$
$$\Delta E_{1,2} = \Delta E_{0,1} - E_c$$

Dal momento che i valori tipici in gioco sono $E_J/\hbar \sim 2GHz$ e $E_c/\hbar \sim 400MHz$ è possibile selezionare sperimentalmente l'unica transizione a cui siamo interessati: quella tra il livello E_0 e E_1; ottenendo così un sistema fisico a due livelli utilizzabile come qubit.

4.3 Oscillazioni Rabi

In questa sezione si vuole fornire la spiegazione teorica di quanto osservato nell'esperimento di caratterizzazione circa le oscillazioni Rabi. Il fenomeno indagato è dunque un sistema a due livelli sotto l'azione continua di radiazione elettromagnetica coerente con la frequenza del sistema [26]. L'approccio risolutivo qui presentato è quello semiclassico.
Assumendo che l'hamiltoniana del qubit sia:

$$\widehat{H}_q = -\omega_q\frac{\sigma_z}{2}$$

dove ω_q rappresenta l'energia necessaria per far transire lo stato da |0> a |1> (per il qubit di tipo trasmon) e σ_z la terza matrice di Pauli.

Per interazione semiclassica, si consideri il momento di dipolo elettrico $\hat{d} = \vec{d}\sigma_x$ del qubit posto in un campo elettrico E oscillante alla sua frequenza di risonanza con dipendenza temporale descritta da $E(t) = Ecos(\omega_d t)$. L'hamiltoniana che descrive l'interazione tra campo e sistema è $H_{int} = -\vec{E}\cdot\hat{d}$;

se si assume, per semplicità, il dipolo elettrico allineato col campo E, allora l'hamiltoniana totale diventa:

$$H_{tot} = -\omega_q \frac{\sigma_z}{2} - Acos(\omega_d t)\sigma_x$$

In cui è stato introdotto il parametro A=Ed che rappresenta quanto è forte il termine d'interazione.
Per conoscere come il sistema evolve è possibile risolvere esplicitamente l'equazione di Schrödinger per hamiltoniana dipendente dal tempo, oppure, cercare la soluzione partendo da un *ansatz*. Infatti, la dipendenza temporale di H_{tot} è data esclusivamente dal termine d'interazione; immaginando ora di "spegnere" il campo elettrico sappiamo che per un generico stato $|\psi>= C_0|0> + C_1|1>$ il suo evoluto temporale sarebbe:

$$|\psi(t)>= C_{0(t)} e^{\,i\frac{\omega_q}{2}t}|0> + C_{1(t)} e^{\,-i\frac{\omega_q}{2}t}|1>$$

dove si è esplicitata la dipendenza temporale per i coefficienti. Assumendo per ansatz questo tipo di soluzione e inserendola nell'equazione di Schrödinger:

$$i\frac{\partial|\psi(t)>}{\partial t} = \left(-\omega_q \frac{\sigma_z}{2} - Acos(\omega_d t)\sigma_x\right)|\psi(t)>$$

Quello che si trova è un sistema di due equazioni differenziali accoppiate per $C_{0(t)}$ e $C_{1(t)}$.

$$\dot{C}_0 = iAcos(\omega_d t) e^{\,-i\omega_q t} C_1$$

$$\dot{C}_1 = iAcos(\omega_d t) e^{\,i\omega_q t} C_0$$

Al fine di risolvere questo analiticamente si espande $cos(\omega_d t) e^{\,\pm i\omega_q t} = e^{\,i(\omega_q \pm \omega_d)t} + e^{\,-i(\omega_q \mp \omega_d)t}$; di cui ignoriamo i termini rapidamente rotanti, ossia $e^{\,\pm i(\omega_q + \omega_d)t}$, in quanto poco interessati alle piccole scale temporali. Dunque, il sistema diventa:

$$\dot{C}_0 = iA e^{\,-i(\omega_q - \omega_d)t} C_1$$

$$\dot{C}_1 = i e^{\,i(\omega_q - \omega_d)t} C_0$$

Si pongano ora le condizioni iniziali ($C_0(0) = 1$, $C_1(0) = 0$) necessarie per risolvere il problema di Cauchy. Così facendo, si ottengono le seguenti evoluzioni temporali per i due coefficienti:

$$C_{0(t)} = \frac{e^{\,-i\frac{\Delta_d}{2}t}}{\Omega_R}\left(\Omega_R cos\left(\frac{\Omega_R}{2}t\right) + i\,\Delta_d sin\left(\frac{\Omega_R}{2}t\right)\right)$$

$$C_{1(t)} = i\,\frac{A\, e^{\,-i\frac{\Delta_d}{2}t}}{\Omega_R}\, sin\left(\frac{\Omega_R}{2}t\right)$$

In cui sono stati inseriti i termini $\Delta_d = \omega_q - \omega_d$ e $\Omega_R = \sqrt{A^2 + {\Delta_d}^2}$ per alleggerire la notazione. Ottenuta l'evoluzione per $|\psi(t)>$ si può ottenere l'evoluzione per qualsiasi osservabile. Al fine di mostrare con maggiore chiarezza quale sia la dinamica del sistema, è comodo osservare l'andamento del popolamento dello stato eccitato:

$$P_{1(t)} = |C_{1(t)}|^2 = \frac{A^2}{{\Omega_R}^2} sin^2\left(\frac{\Omega_R}{2} t\right)$$

Il che giustifica l'andamento oscillatorio osservato sperimentalmente.

A.4 Programma di Caratterizzazione

Nelle pagine seguenti il programma in linguaggio Python usato per la caratterizzazione del qubit del backend ibmq_armonk. Il programma si può eseguire tranquillamente da terminale, previa creazione all'IBM Quantum Experience e necessita di connessione internet, in quanto invia i circuiti da eseguire al cloud IBM. È consigliabile, però, trasporre il programma su uno *jupyter notebook* per seguire *step by step* i passi dell'esecuzione.

```
from qiskit import *
from qiskit.tools.jupyter import *
from qiskit import IBMQ
IBMQ.load_account()
from qiskit.tools.jupyter import *provider = IBMQ.get_provider(hub='ibm-q',
                                                group='open', project='main')

backend = provider.get_backend('ibmq_armonk')
backend_config = backend.configuration()
assert backend_config.open_pulse, "Backend doesn't support Pulse"
dt = backend_config.dt
print(f"Sampling time: {dt*1e9} ns")    # The configuration returns dt in seconds,
                                        # so multiply by 1e9 to get nanoseconds
backend_defaults = backend.defaults()

import numpy as np

GHz = 1.0e9 # Gigahertz
MHz = 1.0e6 # Megahertz
us = 1.0e-6 # Microseconds
ns = 1.0e-9 # Nanoseconds

qubit = 0   # qubit che studiamo
mem_slot = 0    # Memory slot

# Centro della finestra di sweep. Frequenza espressa in Hertz
center_frequency_Hz = backend_defaults.qubit_freq_est[qubit]

print(f"Qubit {qubit} has an estimated frequency of{center_frequency_Hz / GHz} GHz.")

# Fattore di scala (per comodità)
scale_factor = 1e-14

# Ampiezza finestra di sweep
frequency_span_Hz = 40 * MHz
# step di cui mi muovo all'interno della finestra
frequency_step_Hz = 1 * MHz

frequency_min = center_frequency_Hz - frequency_span_Hz / 2
frequency_max = center_frequency_Hz + frequency_span_Hz / 2

# Vettore np delle freq. dell'esperimento. Definisco primo,
#ultimo numero del vettore e la spaziatura tra i valori
frequencies_GHz = np.arange(frequency_min / GHz, frequency_max / GHz,
                            frequency_step_Hz / GHz)

print(f"The sweep will go from {frequency_min / GHz} GHz to {frequency_max / GHz}
                                GHz \in steps of {frequency_step_Hz / MHz} MHz.")

# Siccome i samples devono essere in numero multiplo di 16, definisco una funzione
def get_closest_multiple_of_16(num):
    return int(num + 8 ) - (int(num + 8 ) % 16)

from qiskit import pulse
from qiskit.circuit import Parameter

# Drive pulse parameters (us = microseconds)
drive_sigma_sec = 0.075 * us                # This determines the actual width
drive_duration_sec = drive_sigma_sec * 8    # This is a truncating parameter, because
```

```
t=drive_duration_sec                          # gaussians don't have a natural
drive_amp = 0.05                              # finite length

# Creare il programma di base
# Start with drive pulse acting on the drive channel
freq = Parameter('freq')
with pulse.build(backend=backend, default_alignment='sequential',
                 name='Frequency sweep') as sweep_sched:
    drive_duration = get_closest_multiple_of_16(pulse.seconds_to_samples(t))
    drive_sigma = pulse.seconds_to_samples(drive_sigma_sec)
    drive_chan = pulse.drive_channel(qubit)
    pulse.set_frequency(freq, drive_chan)
    # Drive pulse samples
    pulse.play(pulse.Gaussian(duration=drive_duration,
                              sigma=drive_sigma,
                              amp=drive_amp,
                              name='freq_sweep_excitation_pulse'), drive_chan)
    # Define our measurement pulse
    pulse.measure(qubits=[qubit], registers=[pulse.MemorySlot(mem_slot)])


# Create the frequency settings for the sweep (MUST BE IN HZ)
frequencies_Hz = frequencies_GHz*GHz

schedules = [sweep_sched.assign_parameters({freq : f}, inplace=False)
                                              for f in frequencies_Hz]
schedules[0].draw() #Argomento backend=backend

num_shots_per_frequency = 1024

job = backend.run(schedules,
                  meas_level=1,
                  meas_return='avg',
                  shots=num_shots_per_frequency)
# Monitoro lo stato del job
from qiskit.tools.monitor import job_monitor
job_monitor(job)

# Recupero i risultati
frequency_sweep_results = job.result(timeout=120) # timeout parameter set to 120s
# Estraggo i risultati e li plotto
import matplotlib.pyplot as plt

sweep_values = []
for i in range(len(frequency_sweep_results.results)):
    # Get the results from the ith experiment
    res = frequency_sweep_results.get_memory(i)*scale_factor
    # Get the results for `qubit` from this experiment
    sweep_values.append(res[qubit])

plt.scatter(frequencies_GHz, np.real(sweep_values), color='black')
plt.xlim([min(frequencies_GHz), max(frequencies_GHz)])
plt.xlabel("Frequency [GHz]")
plt.ylabel("Measured signal [a.u.]")
plt.show()

from scipy.optimize import curve_fit

def fit_function(x_values, y_values, function, init_params):
    fitparams, conv = curve_fit(function, x_values, y_values, init_params)
    y_fit = function(x_values, *fitparams)
```

```
    return fitparams, y_fit
fit_params, y_fit = fit_function(frequencies_GHz,
                                 np.real(sweep_values),
                                 lambda x, A, q_freq, B, C: (A / np.pi) *
                                           (B / ((x - q_freq)**2 + B**2)) + C,
                                 [5, 4.975, 1, -15] #initial parameters for curve_fit
                                )
plt.scatter(frequencies_GHz, np.real(sweep_values), color='black')
plt.plot(frequencies_GHz, y_fit, color='red')
plt.xlim([min(frequencies_GHz), max(frequencies_GHz)])

plt.xlabel("Frequency [GHz]")
plt.ylabel("Measured Signal [a.u.]")
plt.show()

A, rough_qubit_frequency, B, C = fit_params
rough_qubit_frequency = rough_qubit_frequency*GHz # make sure qubit freq is in Hz
print(f"We've updated our qubit frequency estimate from "
      f"{round(backend_defaults.qubit_freq_est[qubit] / GHz, 5)} GHz to
                    {round(rough_qubit_frequency/GHz, 5)} GHz.")

#CALIBRATING AND USING A PI_PULSE ( 3 )
# Rabi experiment parameters
num_rabi_points = 50

# Drive amplitude values to iterate over: 50 amplitudes evenly spaced from 0 to 0.75
drive_amp_min = 0
drive_amp_max = 0.75
drive_amps = np.linspace(drive_amp_min, drive_amp_max, num_rabi_points)

#ESPERIMENTO: invio impulsi al qubit alla sua freq. di risonanza, effettuo una misura
# e itero il procedimento per diverse ampiezze di segnale
drive_amp = Parameter('drive_amp')
with pulse.build(backend=backend, default_alignment='sequential',
                            name='Rabi Experiment') as rabi_sched:
    drive_duration = get_closest_multiple_of_16(pulse.seconds_to_samples(drive_duration_se
    drive_sigma = pulse.seconds_to_samples(drive_sigma_sec)
    drive_chan = pulse.drive_channel(qubit)
    pulse.set_frequency(rough_qubit_frequency, drive_chan)
    pulse.play(pulse.Gaussian(duration=drive_duration,
                              amp=drive_amp,
                              sigma=drive_sigma,
                              name='Rabi Pulse'), drive_chan)
    pulse.measure(qubits=[qubit], registers=[pulse.MemorySlot(mem_slot)])

rabi_schedules = [rabi_sched.assign_parameters({drive_amp: a},
                            inplace=False) for a in drive_amps]
rabi_schedules[-1].draw(backend=backend)

num_shots_per_point = 1024
job = backend.run(rabi_schedules,
                  meas_level=1,
                  meas_return='avg',
                  shots=num_shots_per_point)

job_monitor(job)
rabi_results = job.result(timeout=120)

# Ottenuti i risultati, li fittiamo per cercare l'ampiezza del segnale necessaria.
# L'ampiezza della sinusoide la frazione di volte in cui un qubit era in |0> o
```

```
# |1> per ogni data ampiezza.

# center data around 0
def baseline_remove(values):
    return np.array(values) - np.mean(values)
rabi_values = []
for i in range(num_rabi_points):
    # Get the results for `qubit` from the ith experiment
    rabi_values.append(rabi_results.get_memory(i)[qubit] * scale_factor)

rabi_values = np.real(baseline_remove(rabi_values))

plt.xlabel("Drive amp [a.u.]")
plt.ylabel("Measured signal [a.u.]")
plt.scatter(drive_amps, rabi_values, color='black') # plot real part of Rabi values
plt.show()

# Ora fitto questi dati con una sinusoide
fit_params, y_fit = fit_function(drive_amps,
                                 rabi_values,
                                 lambda x, A, B, drive_period, phi:
                                 (A*np.cos(2*np.pi*x/drive_period - phi) + B),
                                 [3, 0.1, 0.3, 0])

plt.scatter(drive_amps, rabi_values, color='black')
plt.plot(drive_amps, y_fit, color='red')

drive_period = fit_params[2] # get period of rabi oscillation

plt.axvline(drive_period/2, color='red', linestyle='--')
plt.axvline(drive_period, color='red', linestyle='--')
plt.annotate("", xy=(drive_period, 0), xytext=(drive_period/2,0),
                    arrowprops=dict(arrowstyle="<->", color='red'))
plt.annotate("$\pi$", xy=(drive_period/2-0.03, 0.1), color='red')

plt.xlabel("Drive amp [a.u.]", fontsize=15)
plt.ylabel("Measured signal [a.u.]", fontsize=15)
plt.show()

pi_amp = abs(drive_period / 2)
print(f"Pi Amplitude = {pi_amp}")

# Definisco il mio impulso in base ai risultati appena ottenuti!!
with pulse.build(backend) as pi_pulse:
    drive_duration = get_closest_multiple_of_16(pulse.seconds_to_samples(t))
    drive_sigma = pulse.seconds_to_samples(drive_sigma_sec)
    drive_chan = pulse.drive_channel(qubit)
    pulse.play(pulse.Gaussian(duration=drive_duration,      #Impulso gaussiano coi
                              amp=pi_amp,                   #parametri ottenuti
                              sigma=drive_sigma,
                              name='pi_pulse'), drive_chan)

# Distinguere tra |0> e |1>. Creo due programmi: uno per lo stato fondamentale e uno
# per lo stato eccitato. Idea: ho trovato come ruotare di pi il mio stato sulla sfera
# di Bloch ora osservo le misure che seguono ad una rotazione per interpretare cosa è
# |0> e cosa invece è |1>
# Create two schedules

# Ground state schedule
with pulse.build(backend=backend, default_alignment='sequential',
                            name='ground state') as gnd_schedule:
```

```
    drive_chan = pulse.drive_channel(qubit)
    pulse.set_frequency(rough_qubit_frequency, drive_chan)
    pulse.measure(qubits=[qubit], registers=[pulse.MemorySlot(mem_slot)])


# Excited state schedule
with pulse.build(backend=backend, default_alignment='sequential',
                           name='excited state') as exc_schedule:
    drive_chan = pulse.drive_channel(qubit)
    pulse.set_frequency(rough_qubit_frequency, drive_chan)
    pulse.call(pi_pulse)
    pulse.measure(qubits=[qubit], registers=[pulse.MemorySlot(mem_slot)])

gnd_schedule.draw(backend=backend)
exc_schedule.draw(backend=backend)

# Execution settings
num_shots = 1024

job = backend.run([gnd_schedule, exc_schedule],
                  meas_level=1,
                  meas_return='single',
                  shots=num_shots)

job_monitor(job)
gnd_exc_results = job.result(timeout=120)

# Mostro i risultati del job nel quale ho preparato i due diversi stati
#per iniziare a distinguerli
gnd_results = gnd_exc_results.get_memory(0)[:, qubit]*scale_factor
exc_results = gnd_exc_results.get_memory(1)[:, qubit]*scale_factor

plt.figure()

# Plot all the results
# All results from the gnd_schedule are plotted in blue
plt.scatter(np.real(gnd_results), np.imag(gnd_results),
                s=5, cmap='viridis', c='blue', alpha=0.5, label='state_0')
# All results from the exc_schedule are plotted in red
plt.scatter(np.real(exc_results), np.imag(exc_results),
                s=5, cmap='viridis', c='red', alpha=0.5, label='state_1')

plt.axis('square')

# Plot a large dot for the average result of the 0 and 1 states.
mean_gnd = np.mean(gnd_results) # takes mean of both real and imaginary parts
mean_exc = np.mean(exc_results)
plt.scatter(np.real(mean_gnd), np.imag(mean_gnd),
            s=200, cmap='viridis', c='black',alpha=1.0, label='state_0_mean')
plt.scatter(np.real(mean_exc), np.imag(mean_exc),
            s=200, cmap='viridis', c='black',alpha=1.0, label='state_1_mean')

plt.ylabel('I [a.u.]', fontsize=15)
plt.xlabel('Q [a.u.]', fontsize=15)
plt.title("0-1 discrimination", fontsize=15)

plt.show()

# Ora creo una funzione Discriminator che meglio distingue i due stati (avendo trovato
# la media per quelli che seguono ad una preparazione di ground o di excited)
# La funzione ritorna 0 se una successiva misura (di qualsiasi altro esperimento/impulso)
```

```
# è più vicina alla media dei punti ottenuti con preparazione di 'ground' e 1 se invece
# più vicino all'altro punto di media.
import math

def classify(point: complex):
    """Classify the given state as |0> or |1>."""
    def distance(a, b):
        return math.sqrt((np.real(a) - np.real(b))**2 + (np.imag(a) - np.imag(b))**2)
    return int(distance(point, mean_exc) < distance(point, mean_gnd))

# Misura del T1, ovvero il tempo di decadimento da stato eccitato a stato di ground
# IDEA: simile a prima -> Applico un pi_pulse ed effettuo una misura MA queste due
# operazioni NON le svolgo immediatamente. Inseriamo tra i due un certo intervallo di
# lunghezza via via crescente e creiamo un grafico che mostra la frazione di misure che
# restituiscono |1> in funzione della durata dell'intervallo.
# Definiamo T1 il tempo caratteristico della decrescita esponenziale osservata.

# T1 experiment parameters
time_max_sec = 450 * us
time_step_sec = 6.5 * us
delay_times_sec = np.arange(1 * us, time_max_sec, time_step_sec)

# Create schedules for the experiment
t1_schedules = []
for delay in delay_times_sec:
    with pulse.build(backend=backend, default_alignment='sequential',
                     name=f"T1 delay = {delay / ns} ns") as t1_schedule:
        drive_chan = pulse.drive_channel(qubit)
        pulse.set_frequency(rough_qubit_frequency, drive_chan)
        pulse.call(pi_pulse)
        pulse.delay(get_closest_multiple_of_16(pulse.seconds_to_samples(delay)),
                    drive_chan)
        pulse.measure(qubits=[qubit], registers=[pulse.MemorySlot(mem_slot)])
    t1_schedules.append(t1_schedule)

sched_idx = 0
t1_schedules[sched_idx].draw(backend=backend)

# Execution settings
num_shots = 256

job = backend.run(t1_schedules,
                  meas_level=1,
                  meas_return='single',
                  shots=num_shots)

job_monitor(job)
t1_results = job.result(timeout=120)

t1_values = []

for i in range(len(delay_times_sec)):
    iq_data = t1_results.get_memory(i)[:,qubit] * scale_factor
    t1_values.append(sum(map(classify, iq_data)) / num_shots)

plt.scatter(delay_times_sec/us, t1_values, color='black')
plt.title("$T_1$ Experiment", fontsize=15)
plt.xlabel('Delay before measurement [$\mu$s]', fontsize=15)
plt.ylabel('Signal [a.u.]', fontsize=15)
plt.show()
```

```
job = backend.run(ramsey_schedules,
                  meas_level=1,
                  meas_return='single',
                  shots=num_shots)

job_monitor(job)
ramsey_results = job.result(timeout=120)

ramsey_values = []

for i in range(len(delay_times_sec)):
    iq_data = ramsey_results.get_memory(i)[:,qubit] * scale_factor
    ramsey_values.append(sum(map(classify, iq_data)) / num_shots)

plt.scatter(delay_times_sec/us, np.real(ramsey_values), color='black')
plt.xlim(0, np.max(delay_times_sec/us))
plt.title("Ramsey Experiment", fontsize=15)
plt.xlabel('Delay between X90 pulses [$\mu$s]', fontsize=15)
plt.ylabel('Measured Signal [a.u.]', fontsize=15)
plt.show()

fit_params, y_fit = fit_function(delay_times_sec/us, np.real(ramsey_values),
                                 lambda x, A, del_f_MHz, C, B: (
                                          A * np.cos(2*np.pi*del_f_MHz*x - C) + B
                                         ),
                                 [5, 1./0.4, 0, 0.25]
                                )

# Off-resonance component
_, del_f_MHz, _, _, = fit_params # freq is MHz since times in us

plt.scatter(delay_times_sec/us, np.real(ramsey_values), color='black')
plt.plot(delay_times_sec/us, y_fit, color='red', label=f"df = {del_f_MHz:.2f} MHz")
plt.xlim(0, np.max(delay_times_sec/us))
plt.xlabel('Delay between X90 pulses [$\mu$s]', fontsize=15)
plt.ylabel('Measured Signal [a.u.]', fontsize=15)
plt.title('Ramsey Experiment', fontsize=15)
plt.legend()
plt.show()

precise_qubit_freq = rough_qubit_frequency + (del_f_MHz - detuning_MHz) * MHz
print(f"Our updated qubit frequency is now {round(precise_qubit_freq/GHz, 6)} GHz. "
      f"It used to be {round(rough_qubit_frequency / GHz, 6)} GHz")

# Measuring T2 using Hanh Echoes
# Simile al Ramsey Experiment: sequenza di impulsi pi/2, pi e pi/2
# Il tempo di decadimento di quest'esperimento retituisce T2: tempo di coerenza

# T2 experiment parameters
tau_max_sec = 200 * us
tau_step_sec = 4 * us
delay_times_sec = np.arange(2 * us, tau_max_sec, tau_step_sec)

t2_schedules = []
for delay in delay_times_sec:
    with pulse.build(backend=backend, default_alignment='sequential',
                     name=f"T2 delay = {delay / ns} ns") as t2_schedule:
        drive_chan = pulse.drive_channel(qubit)
        pulse.set_frequency(precise_qubit_freq, drive_chan)
        pulse.call(x90_pulse)
        pulse.delay(get_closest_multiple_of_16(pulse.seconds_to_samples(delay)),
```

```
                    drive_chan)
        pulse.call(pi_pulse)
        pulse.delay(get_closest_multiple_of_16(pulse.seconds_to_samples(delay)),
                    drive_chan)
        pulse.call(x90_pulse)
        pulse.measure(qubits=[qubit], registers=[pulse.MemorySlot(mem_slot)])
    t2_schedules.append(t2_schedule)
t2_schedules[-1].draw(backend=backend)

# Execution settings
num_shots_per_point = 512

job = backend.run(t2_schedules,
                  meas_level=1,
                  meas_return='single',
                  shots=num_shots_per_point)

job_monitor(job)
t2_results = job.result(timeout=120)

t2_values = []
for i in range(len(delay_times_sec)):
    iq_data = t2_results.get_memory(i)[:,qubit] * scale_factor
    t2_values.append(sum(map(classify, iq_data)) / num_shots_per_point)

plt.scatter(2*delay_times_sec/us, t2_values, color='black')
plt.xlabel('Delay between X90 pulse and $\pi$ pulse [$\mu$s]', fontsize=15)
plt.ylabel('Measured Signal [a.u.]', fontsize=15)
plt.title('Hahn Echo Experiment', fontsize=15)
plt.show()

fit_params, y_fit = fit_function(2*delay_times_sec/us, t2_values,
             lambda x, A, B, T2: (A * np.exp(-x / T2) + B),
             [-3, 0, 100])

_, _, T2 = fit_params
print()

plt.scatter(2*delay_times_sec/us, t2_values, color='black')
plt.plot(2*delay_times_sec/us, y_fit, color='red', label=f"T2 = {T2:.2f} us")
plt.xlim(0, np.max(2*delay_times_sec/us))
plt.xlabel('Delay between X90 pulse and $\pi$ pulse [$\mu$s]', fontsize=15)
plt.ylabel('Measured Signal [a.u.]', fontsize=15)
plt.title('Hahn Echo Experiment', fontsize=15)
plt.legend()
plt.show()
```

Bibliografia

[1] Michael A. Nielsen, Isaac L. Chuang. *Quantum Computation and Quantum Information*, 10th Anniversary edition. 2010.

[2] Stephen Gasiorowicz. *Quantum Physics, 3nd edition*. 2003.

[3] David P. DiVincenzo. *The Physical Implementation of Quantum Computation*. https://arxiv.org/abs/quant-ph/0002077, 2000.

[4] David P. Pappas. *Fabrication of superconducting circuits and a universal gate set for strongly ZZ coupled qubits,* Univeristà degli studi Milano-Bicocca. 4 Aprile 2021.

[5] Google Quantum AI. https://quantumai.google/.

[6] Arute, F., Arya, K., Babbush, R. *et al. Quantum supremacy using a programmable superconducting processor. Nature* 2019. https://doi.org/10.1038/s41586-019-1666-5, 2019.

[7] Qiskit. https://qiskit.org/textbook-beta/.

[8] National Academies of Sciences, Engineering, and Medicine. *Quantum Computing: Progress and Prospects*. Washington: The National Academies Press. 2019. https://doi.org/10.17226/25196.

[9] Pengfei Wang, Chun-Yang Luan, Mu Qiao et al. *Single ion-qubit exceeding one hour coherence time.* arXiv:2008.00251v1*,* 2020.

[10] IBM. https://research.ibm.com/blog/ibm-quantum-roadmap.

[11] Stefano Olivares. *Lecture Notes on Quantum Computing.* 4th edition. 2020.

[12] IBM Quantum. https://quantum-computing.ibm.com.

[13] IBM Quantum Services. https://quantum-computing.ibm.com/services.

[14] Qiskit. https://qiskit.org/.

[15] QiskitPulse. https://qiskit.org/documentation/apidoc/pulse.html.

[16] Stefania Balasiu. *Single-Qubit Gates Calibration in PycQED using Superconducting Qubits.* 2017.

[17] Easwar Magesan, Jay M. Gambetta, Joseph Emerson. *Characterizing Quantum Gates via Randomized Benchmarking.* https://arxiv.org/abs/1109.6887. 2012.

[18] Felix Motzoi, Lukas Buchmann. *Simple, smooth and fast pulses for dispersive measurements in cavities and quantum networks,* https://arxiv.org/abs/1809.04116. 2018.

[19] Qiskit. https://qiskit.org/documentation/stubs/qiskit.compiler.transpile.html.

[20] Adrian Perez-Salinas, Alba Cervera-Lierta, Elies Gil-Fuster, Josè I. Latorre. *Data re-uploading for a universal quantum classifier.* arXiv:1907.02085v2. 2020.

[21] EAGE. https://www.eage.it/machine-learning/classificazione-machine-learning.

[22] George Cybenko. *Approximation by superpositions of a sigmoidal function* https://doi.org/10.1007/BF02551274. 1989.

[23] Qibo. https://arxiv.org/abs/2009.01845.

[24] Qibo. https://doi.org/10.5281/zenodo.3997194.

[25] Andrea Giachero, Federico Galizzi. Github repository: https://github.com/qismib/PIQE. 2021.

[26] Mahdi Naghiloo. *Introduction to Experimental Quantum Measurement with Superconducting Qubits.* https://arxiv.org/abs/1904.09291. 2019

[27] Paolo Villoresi. *Quantum Comuncations in Space enabling new tests of Quantum Mechanics and secure communications,* Univeristà degli studi Milano-Bicocca. 9 Aprile 2021.

[28] Marco Genovese. *Quantum Imaging: una breve introduzione,* Univeristà degli studi Milano Bicocca. 24 Maggio 2021.