
Università degli Studi di Milano-Bicocca

Dipartimento di Fisica

Corso di laurea triennale in Fisica

# Sviluppo di un algoritmo variazionale per la dinamica dei sistemi quantistici

**Relatore:**
Andrea Giachero

**Correlatore:**
Stefano Barison

**Candidato:**
Francesco Paganelli

**Anno Accademico 2022/2023**

# Sommario

In questa tesi verrà studiato un algoritmo variazionale per la simulazione di sistemi dinamici su computer quantistici noto come *projected-Variational Quantum Dynamics* (pVQD).

Il primo capitolo vuole racchiudere la teoria di base della computazione quantistica. In una prima parte (Sezione 1.1), si introducono i concetti fondamentali della computazione quantistica. Si comincia dall'unità base dell'informazione, con la relativa formalizzazione matematica, fino alle operazioni che si possono compiere. Successivamente, in un'ottica più pratica, si mostra come attualmente si programma computer quantistici dando una breve introduzione di Qiskit, pacchetto Python che permette di accedere ai computer quantistici di IBM. Si continua (Sezione 1.2) con una descrizione del rumore quantistico, della sua rappresentazione matematica e delle strategie per affrontarlo e mitigarlo. Si conclude (Sezione 1.3) con l'introduzione teorica agli algoritmi di tipo variazionale.

Nel secondo capitolo, si descrive la specifica teoria che riguarda l'obbiettivo della tesi, presentando (Sezione 2.1) l'idea della simulazione della dinamica su computer quantistici, la costruzione di un simulatore ideale ed un semplice esempio pratico. Poi (Sezione 2.2) si introducono gli algoritmi variazionali utilizzati nel campo della simulazione quantistica, concludendo (Sezione 2.3) con la descrizione specifica del funzionamento del pVQD.

Il terzo capitolo è dedicato alla simulazione di una catena di spin interagenti secondo il modello di Ising. Inizialmente (Sezione 3.1), si descrive tale modello e gli specifici parametri da applicare al pVQD. Infine (Sezione 3.2), vengono presentati tutti i risultati ottenuti,sia usando un simulatore ideale che introducendo modelli d'errore.

Si conclude con una breve analisi dei risultati e uno sguardo su quelle che potrebbero essere le strategie da applicare perché l'algoritmo possa essere usato anche su hardware reale.

# Indice

**Sommario** **III**

**Introduzione** **IX**

**1 Teoria della computazione quantistica** **1**
1.1 Computer quantistici, qubit e circuiti . . . . . 1
1.1.1 Concetti fondamentali . . . . . 1
1.1.2 Operazioni su qubit . . . . . 3
1.1.3 Traduzione di operatori unitari in gate elementari . . 6
1.1.4 Misura . . . . . 8
1.1.5 Circuiti . . . . . 8
1.1.6 Qiskit . . . . . 9
1.2 Quantum noise . . . . . 11
1.2.1 Noisy Intermediate-Scale Quantum devices . . . . . 11
1.2.2 Error mitigation . . . . . 13
1.2.3 Descrizione matematica del rumore: operatore densità 16
1.2.4 Modelli di rumore . . . . . 19
1.3 Variational Quantum Algorithms . . . . . 23
1.3.1 Struttura di un VQA . . . . . 23

**2 Simulation of quantum dynamics** **29**
2.1 Universal quantum simulation . . . . . 29
2.1.1 Struttura di un simulatore . . . . . 29
2.1.2 Esempio: evoluzione dinamica tramite Trotter . . . . 32
2.2 Variational quantum simulation . . . . . 33
2.2.1 Principi variazionali classici . . . . . 34
2.2.2 Principi variazionali quantistici . . . . . 36
2.3 Projected Variational Quantum Dynamics . . . . . 36
2.3.1 Ansatz e cost function . . . . . 37
2.3.2 Ottimizzazione . . . . . 40

**3 Evoluzione dinamica del modello di Ising** **43**
3.1 Modello di evoluzione . . . . . 43
3.1.1 Ansatz . . . . . 43

3.1.2 Trotter step . . . . . 44
3.1.3 Misura delle osservabili . . . . . 44
3.2 Risultati . . . . . 45
3.2.1 Simulatori ideali . . . . . 45
3.2.2 Simulatori con rumore . . . . . 50

**4 Conclusioni** **57**
4.1 Considerazioni sui risultati . . . . . 57
4.2 Prospettive future . . . . . 58

# Elenco delle figure


1.1 Sfera di Bloch. . . . . 2
1.2 Circuito per $R_{zz}$. . . . . 7
1.3 Circuito per $R_{yy}$. . . . . 7
1.4 Circuito per $R_{xz}$. . . . . 8
1.5 La principale notazione utilizzata del disegnare diagrammi di circuiti. . . . . 9
1.6 Diagramma del circuito eseguito. . . . . 11
1.7 Risultati della misura del circuito con un simulatore senza rumore e sul computer quantistico "ibmq_quito". . . . . 12
1.8 Rappresentazione dei due passi dello ZNE. . . . . 14
1.9 Rappresetazione schematica della PEC, gli operatori reali possono essere schematizzati come la quasi-probability distribution di opearatori reali (o viceversa), scoprendo il legame tra i due si può costruire un circuito che cancelli il rumore. . . . 15
1.10 Diagramma, tratto da [14], del funzionamento dei VQA. . . . 24

2.1 Schema del funzionamento di un Universal Quantum Simulator, figura tratta da [6]. . . . . 31
2.2 Evoluzione di un sistema di tre spin sotto il modello di Ising al variare del numero di trotter steps. . . . . 33
2.3 Circuito per valutare l'Equazione 2.5. . . . . 38

3.1 Circuito che implementa l'ansatz con una profondità 1 per 3 spin. Aumentare la profondità significa ripetere $d$ volte questo set di gate. Si noti anche l'implementazione di $R_{zz}(\theta)$ tramite i due CNOT (Sezione 1.1.3). . . . . 44
3.2 Circuito che implementa il singolo Trotter step. . . . . 45
3.3 Confronto tra la simulazione tramite statevector (senza shot), la simulazione con shots ed il valore atteso. . . . . 46
3.4 Confronto tra step di ottimizzazione richiesti nel caso in cui si parta dallo shift precedente oppure si parta sempre da uno shift nullo. L'ottimizzazione è stata eseguita tramite GD. . . 47
3.5 Step di ottimizzazione per GD, ADAM e SPSA. . . . . 48

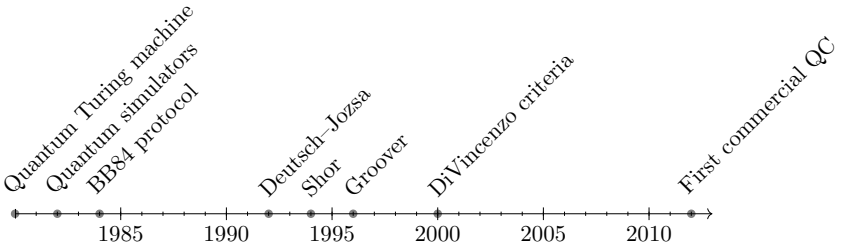
3.6 Curva di ottimizzazione con i vari ottimizzatori, si noti che l'asse delle y è in scala logaritmica. In alto $t = 0.05$, mentre in basso $t = 1.5$. . . . . 49
3.7 Confronto tra gli step di ottimizzazione necessari con la cost function locale e quella globale. . . . . 50
3.8 Curve di ottimizzazione per $t = 0.05$ a diverse intensità di rumore. L'ottimizzatore utilizzato è il GD. A destra in scala normale, a sinistra in scala logaritmica. . . . . 51
3.9 Valore minimo della cost function globale ottimizzata in modo esatto, ma misurata in presenza di rumore. A causa della scala del grafico sembra che la $L$ esatta sia nulla, in realtà è solo minore di $10^{-5}$. . . . . 52
3.10 Cost function al variare degli shift a diverse intensità di rumore. Sopra a $t = 0.05$, sotto a $t = 0.3$. . . . . 53
3.11 Confronto tra le misure con e senza ZNE a diverse intensità di rumore. . . . . 54
3.12 Simulazione in presenza di rumore con ZNE. . . . . 55

# Introduzione

Con *computer quantistico* si intende una macchina che sfrutta le leggi della meccanica quantistica per processare le informazioni e compiere operazioni logiche. Il primo a fornire un modello matematico alla computazione quantistica fu Paul Benioff [1], che nel 1980 introdusse la Quantum Turing machine. Poco dopo, nel 1982, Richard Feynman suggerì di utilizzare questi computer per simulare efficientemente sistemi quantistici [2]. Da qui la ricerca in questo campo divenne mano a mano più popolare, in quanto prometteva di poter raggiungere obbiettivi inimmaginabili alla computazione classica. Nel 1992 venne pubblicato l'algoritmo di Deutsch–Jozsa [3] che risolve un problema senza utilità pratica, ma per cui non esiste soluzione classica efficiente. La più grande svolta si ebbe con la pubblicazione dell'algoritmo di Shor [4]. Nel 1994, Peter Shor sfruttò le peculiarità della computazione quantistica per fattorizzare un intero in numeri primi, un problema di cui non si conosce soluzione efficiente su computer classici e su cui si basano gli algoritmi di crittografia odierni.

In pochi anni si arriva alla la costruzione dei primi hardware quantistici. Ogni computer di questo tipo deve soddisfare i criteri di DiVincenzo [5], ovvero le condizioni necessarie perché un quantum computer sia in grado di simulare efficientemente sistemi quantistici. Sono state proposte diverse tecnologie per la realizzazione fisica di queste macchine, ma le più promettenti sono due [6]: i circuiti superconduttori o a trappola di ioni. La prima tecnologia è utilizzata, ad esempio, nei computer di IBM [7] e di Google [8] e sfrutta lo spettro energetico delle cariche o delle correnti in un superconduttore [6]. La seconda tecnologia, sfruttata ad esempio nei computer di IonQ [9] e Quantinuum [10], utilizza dei campi elettrici per isolare degli ioni.

L'obbiettivo è quello di raggiungere il *quantum advantage* [11], ossia la superiorità pratica rispetto ai computer classici in determinate task. Come dimostrato dagli algoritmi prima citati, nella teoria questo obbiettivo sembra realizzabile, ma ci sono una serie di problemi tecnici da risolvere. Tutti i computer quantistici attualmente esistenti hanno ancora un'affidabilità limitata, ed è per questo che son detti *Noisy Intermediate-Scale Quantum devices* (NISQ) [12]. Su questo tipo di macchine gli algoritmi di Shor e di Deutsch–Jozsa non sono ancora in grado di essere eseguiti su larga scala. Siccome l'era dei computer quantistici performanti sembra ancora lontana,




per raggiungere il quantum advantage si sta facendo affidamento, con buoni risultati [13], a algoritmi alternativi che coinvolgono sia la computazione classica che quella quantistica. Queste tecniche sono note come *Variational Quantum Algorithm* [14, 15]. Tra questi vi sono il *Quantum Aproximate Optimization Algorithm* (QAOA) [16], per risolvere problemi di ottimizzazione, il *Variational Quantum Eigensolver* (VQE) [17], per trovare lo stato di ground di sistemi quantistici o il *projected-Variational Quantum Dynamics* (pVQD) [18], per la simulazione dinamica.

Oggetto di questa tesi sarà lo studio del pVQD. La prima parte dell'analisi riguarderà il funzionamento ideale dell'algoritmo, per poi studiarlo in situazioni più realistiche, nell'ottica di trovare le migliori strategie per poterlo utilizzare su devices NISQ già esistenti attualmente.

# Capitolo 1

# Teoria della computazione quantistica

Il seguente capitolo serve da introduzione teorica ai problemi affrontati durante la tesi. Si comincia (Sezione 1.1) con alcuni concetti di base della computazione quantistica, come i qubit, circuiti quantistici, gate e la programmazione su computer quantistici, in una trattazione basata principalmente su [19, 20, 21]. Quindi (Sezione 1.2), si affronta il maggior problema che si incontra quando si utilizzano i computer quantistici attuali, il rumore, dandone una descrizione matematica ed esponendo algoritmi che per trovarne rimedio. Si conclude (Sezione 1.3) con un'introduzione all'utilizzo di algoritmi variazionali quantistici.

## 1.1 Computer quantistici, qubit e circuiti

### 1.1.1 Concetti fondamentali

La differenza fondamentale tra la computazione classica e quella quantistica sta nella codifica dell'informazione, nel primo caso viene trasportata da *bit* e nel secondo da bit quantistici, i *qubit*. Il primo può trovarsi unicamente nello stato 0 o nello stato 1, come una moneta che assume il valore testa o croce, mentre il secondo si può trovare in un numero infinito di stati, il che viene descritto matematicamente come la combinazione lineare a coefficienti complessi degli stati $|0\rangle$ e $|1\rangle$:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \tag{1.1}$$

L'interpretazione fisica di tale concetto è la *sovrapposizione di stati*, per la quale nell'istante in cui un qubit viene misurato, il suo stato viene proiettato con probabilità $|\alpha|^2$ su $|0\rangle$ e con probabilità $|\beta|^2$ su $|1\rangle$, dalla definizione di probabilità consegue che

$$|\alpha|^2 + |\beta|^2 = 1 \tag{1.2}$$

È bene anche notare che esiste una certa arbitrarietà nella descrizione matematica del qubit, la *base computazionale* $\{|0\rangle, |1\rangle\}$ non è l'unica utilizzabile, ma ne esiste un numero infinito, tra le più comuni vi è $\{|+\rangle, |-\rangle\}$, detta *X-basis*, dove

$$|+\rangle \equiv \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \qquad |-\rangle \equiv \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \tag{1.3}$$

Quella dell'Equazione 1.1 e dell'Equazione 1.3, è detta rappresentazione tramite *bra-ket*, ma non è l'unica possibile. Di seguito alcune tra le più comuni. Nell'Equazione 1.4, viene utilizzata quella vettoriale; una volta scelti due vettori ortonormali arbitrari per scrivere la base, tipicamente

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

si costruisce lo stato usando la definizione di somma vettoriale e prodotto per scalare:

$$\alpha\,|0\rangle + \beta\,|1\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \tag{1.4}$$

Si noti ora che, per le proprietà dei numeri complessi, non si perde di generalità se si applica la sostituzione $\alpha = \cos\theta/2$ e $\beta = e^{i\varphi}\sin\theta/2$, lo stato $|\psi\rangle$ può dunque essere visto come raggio vettore su una sfera unitaria, detta *sfera di Bloch*, come mostrato in Figura 1.1. Più avanti (Sezione 1.2.3) verrà mostrata una generalizzazione di questa rappresentazione, la *density matrix*.

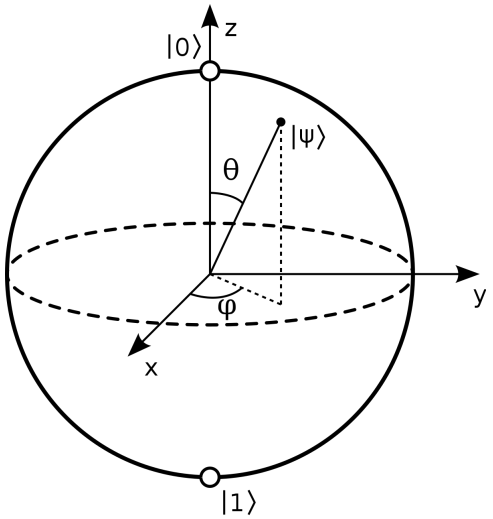



Figura 1.1: Sfera di Bloch.

Così come un computer classico non utilizza un singolo bit, anche in quelli quantistici viene utilizzata una combinazione di più qubit, per descriverli si utilizza il concetto di *prodotto tensoriale* ($\otimes$) tra spazi di Hilbert.

Si prenda un sistema classico di due bit, tutte le possibili combinazioni di valori sono, "00, 01, 10 e 11". Nel momento in cui si misurano due qubit, anche questi possono assumere solo tali valori, che quindi corrisponderanno agli elementi della base computazionale su cui viene scritta la combinazione lineare che descrive i qubit. Proprio quello che si ottiene tramite il prodotto tensoriale $(\alpha' |0\rangle + \beta' |1\rangle) \otimes (\gamma' |0\rangle + \delta' |1\rangle)$:

$$|\psi\rangle = \alpha |00\rangle + \beta |01\rangle + \gamma |10\rangle + \delta |11\rangle$$

sempre con la condizione che la norma di $|\psi\rangle$ sia 1 (Equazione 1.2). In generale, uno stato di $n$ qubit, viene indicato tramite

$$\psi = \frac{\sum_i a_i |i\rangle}{\sqrt{\sum_i |a_i|^2}} \tag{1.5}$$

dove $i \in \{|00\ldots0\rangle, \ldots, |11\ldots1\rangle\}$[1], e il denominatore rappresenta la condizione di normalizzazione.

### 1.1.2 Operazioni su qubit

Classicamente le operazioni sui bit vengono svolte da *gate*, per un solo bit l'unico gate non banale è il NOT, l'operazione per cui $0 \to 1$ e $0 \to 1$, mentre per due bit i gate principali sono AND e OR, descritti dalle *truth table* in Tabella 1.1.

| input | output |
|---|---|
| 00 | 1 |
| 01 | 0 |
| 10 | 0 |
| 11 | 1 |

(a) Tabella logica per il gate AND

| input | output |
|---|---|
| 00 | 0 |
| 01 | 1 |
| 10 | 1 |
| 11 | 1 |

(b) Tabella logica per il gate OR

Tabella 1.1: Tabelle logiche per i gate classici a due bit AND e OR

Quelli appena citati non sono gli unici gate a due bit possibili, ma ne esiste uno per ogni combinazione di truth table che si possa creare. Ci saranno poi anche gate a 3 o più bit. A prima vista sembrerebbe dunque impossibile costruire una macchina in grado di compiere ogni operazione, perché servirebbe un numero infinito di gate diversi, ognuno che corrisponde a una certa tabella di verità. Fortunatamente, si può dimostrare l'esistenza degli *universal gate set*, ovvero insiemi finiti di gate che possono essere combinati per costruire un qualsiasi gate con un numero arbitrario di bit. Il set più famoso è {AND, OR, NOT}.

[1] La base ha dimensione $2^n$

Quanto detto fino ad ora può essere esteso anche a operazioni su qubit. Nella meccanica quantistica un'operazione sullo stato $|\psi\rangle$ è compiuta da un operatore unitario[2] $\hat{U}$, cioè tale per cui

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

Tutte le operazioni sui qubit sono compiute da *gate quantistici* che devono essere operatori unitari di dimensione $2^n \times 2^n$, dove $n$ è il numero di qubit.

Nella computazione quantistica i gate a singolo qubit sono infiniti, a differenza di quanto avviene classicamente, tra i più comuni si hanno i seguenti:

- **Gate identità.** Indicato con $I$, ha un corrispettivo classico, si tratta di un gate banale che mappa lo stato in se stesso, $\hat{I}|\psi\rangle = |\psi\rangle$, la sua matrice è
$$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

- **NOT gate.** Indicato con $X$, ha un corrispettivo classico, inverte le ampiezze degli elementi della base, cioè mappa $|0\rangle$ in $|1\rangle$ e viceversa. La matrice che lo rappresenta è
$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$
e corrisponde ad una rotazione di $\pi$ attorno all'asse x nella sfera di Bloch (Figura 1.1).

- **Z gate.** Indicato con $Z$, non ha un corrispettivo classico, cambia il segno del coefficiente di $|1\rangle$, corrisponde ad una rotazione di $\pi$ attorno all'asse z ed ha matrice
$$Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

- **Y gate.** Indicato con $Y$, non ha un corrispettivo classico, si tratta di una rotazione attorno all'asse y di $\pi$, ed è la mappa $|0\rangle \mapsto i|1\rangle, |1\rangle \mapsto -i|0\rangle$. Ha matrice
$$Y = \begin{pmatrix} 0 & i \\ -i & 0 \end{pmatrix}$$

- **Hadamard gate.** Indicato con $H$, non ha il corrispettivo classico, corrisponde al cambiamento di base $\{|0\rangle, |1\rangle\}$ in $\{|+\rangle, |-\rangle\}$, come già definiti nell'Equazione 1.3. Ha matrice
$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

[2] Le trasformazioni unitarie conservano la norma.

e corrisponde a due rotazioni consecutive sulla sfera di Bloch, di $\pi/2$ attorno a y, quindi di $\pi$ attorno a x. Si noti il fattore di normalizzazione $1/\sqrt{2}$, necessario affinchè la matrice sia unitaria.

- **Phase gate.** Indicato con $S$, non ha un corrispettivo classico, agisce lasciando $|0\rangle$ invariato e aggiungendo una fase $i$ a $|1\rangle$, ha matrice

$$S = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}$$

corrisponde ad una rotazione di $\pi/2$ attorno all'asse $Z$, ciò significa anche $S^2 = Z$.

- **$\frac{\pi}{8}$ gate.** Indicato con $T$, non ha un corrispettivo classico, lascia $|0\rangle$ invariato ed aggiunge una fase $e^{i\pi/4}$ a $|1\rangle$. La sua matrice è

$$T = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix}$$

e corrisponde a una rotazione di $\pi/4$ attorno all'asse $Z$, da cui $T^2 = S$ e $T^4 = Z$.

I gate $X$, $Y$ e $Z$, non sono altro che le matrici di Pauli, quindi sono anche detti *Pauli gates*. Si osservi che si tratta sempre di gate *reversibili*, significa che conoscendo l'output si può sempre ricavare l'input, si tratta di una proprietà valida per ogni gate quantistico, ed è dovuta all'invertibilità delle matrici unitarie. Viceversa gate classici non sono necessariamente invertibili, come esempio si veda il gate AND, Tabella 1.1. Si può dimostrare che i gate classici reversibili sono anche gate quantistici [20].

Come già accennato, i gate a singolo qubit sono infiniti, ognuno corrispondente a una diversa rotazione sulla sfera di Bloch, risulta quindi estremamente utile il fatto che si possano scrivere le rotazioni attorno agli assi $x$, $y$ e $z$ usando solo i gate di Pauli. Come mostrato in [19, 20, 6] il più generico gate a singolo qubit, cioè la generica rotazione di $\theta$ attorno all'asse $\hat{n} = (n_x, n_y, n_z)$ è

$$U = R_n(\theta) = \cos\frac{\theta}{2}I - i\sin\frac{\theta}{2}(n_x X + n_y Y + n_z Z) \tag{1.6}$$

Per quanto riguarda gate a due qubit il più importante è il CNOT, o *controlled not*, si tratta della generalizzazione del gate X; uno dei due qubit è detto *di controllo*, mentre l'altro è il *target*, se il qubit di controllo è 1, allora viene applicato un NOT al secondo, se invece è 0, allora non viene applicata alcuna trasformazione. Gli elementi della base trasformano come

segue:

$$|00\rangle \mapsto |00\rangle$$
$$|01\rangle \mapsto |01\rangle$$
$$|10\rangle \mapsto |11\rangle$$
$$|11\rangle \mapsto |10\rangle$$

La matrice che descrive il CNOT è la matrice $4 \times 4$:

$$\text{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Si consideri il sistema di 2 qubit $|0\rangle \otimes |0\rangle$, se si applicano due generici gate a singolo qubit si ha $U_1 |0\rangle \otimes U_2 |0\rangle$. In tal caso si scrive $(U_1 \otimes U_2)(|0\rangle \otimes |0\rangle)$. In generale, in un sistema di $n$ qubit l'operazione totale è descritta matematicamente come un prodotto tensoriale tra le matrici che rappresentano tutti i gate. Ad esempio, preso un circuito di due qubit con X che agisce sul secondo, si ha

$$(I \otimes X) = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{1.7}$$

Anche nel caso quantistico esistono set universali di gate, alcuni di questi sono:

- $\{CNOT, SU(2)\}$ [22], dove con $SU(2)$ si intende l'insieme delle matrici unitarie speciali[3] $2 \times 2$. Di fatto sono le rotazioni in Equazione 1.6, ovvero tutti i gate a singolo qubit.
- $\{CNOT, H, T\}$ [23]. A differenza dell'insieme precedente, con questo set non si ottengono operatori unitari esatti, ma è possibile approssimarli con un grado di precisione arbitrario.

### 1.1.3 Traduzione di operatori unitari in gate elementari

Tra gli universal gate set più utilizzati vi è $\{CNOT, SU(2)\}$, torna dunque utile capire come ricavare un generico gate a due qubit a partire da questo insieme. Si supponga di avere a disposizione un computer quantistico in grado di implementare CNOT e rotazioni al singolo asse ($\alpha = X, Y, Z$), cioè

$$R_\alpha(\theta) = e^{-i\frac{\theta}{2}\sigma_\alpha} = \cos\frac{\theta}{2} I - i \sin\frac{\theta}{2}\sigma_\alpha \tag{1.8}$$

[3] A determinante 1.

dove $\sigma_\alpha$ sono le matrici di Pauli.

A partire da questi operatori è possibile ricavare la rotazione di un certo angolo $\theta$ di due qubit attorno a un asse, operatore indicato come $R_{\alpha\beta}(\theta)$, con $\alpha, \beta = X, Y, Z$. Per ottenerle si osservi che $R_{zz}(\theta)$ ha struttura

$$R_{zz}(\theta) = e^{-i\frac{\theta}{2}Z\otimes Z} = \cos\theta I \otimes I - i\sin\theta Z \otimes Z$$

corrispondente alla matrice

$$R_{zz} = \begin{pmatrix} \cos\theta - i\sin\theta & 0 & 0 & 0 \\ 0 & \cos\theta + i\sin\theta & 0 & 0 \\ 0 & 0 & \cos\theta + i\sin\theta & 0 \\ 0 & 0 & 0 & \cos\theta - i\sin\theta \end{pmatrix}$$

ottenuta calcolando i prodotti tensoriali tra le matrici. Ricordando quanto detto alla fine della Sezione 1.1.2, la rotazione $R_z$ che agisce sul singolo qubit, in un sistema di due, è il prodotto $I \otimes Z$, osservandone la matrice si nota che è sufficiente scambiare i segni degli ultimi due termini della diagonale per ricavare quella di $R_{zz}$. Il che può essere ottenuto applicando prima e dopo un CNOT, risulta che

$$R_{zz} = CNOT(I \otimes R_z)CNOT$$

ovvero il circuito in Figura 1.2.

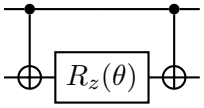



Figura 1.2: Circuito per $R_{zz}$.

Questo è il punto di partenza per ricavare i termini generati dalle rotazioni $\sigma_\alpha \otimes \sigma_\beta$, per cui è sufficiente cambiare frame di riferimento [6]. In particolare si noti che

$$R_y\left(\frac{\pi}{2}\right)\sigma_z R_y\left(-\frac{\pi}{2}\right) = \sigma_x \qquad R_x\left(\frac{\pi}{2}\right)\sigma_z R_x\left(-\frac{\pi}{2}\right) = -\sigma_y$$

perciò una volta applicato $R_{zz}$, se si volesse calcolare $R_{yy}$, bisognerà applicare la seconda relazione ad entrambi i qubit (Figura 1.3), se invece si volesse trovare $R_{xz}$ bisognerà applicare la prima relazione solo al primo qubit (Figura 1.4).

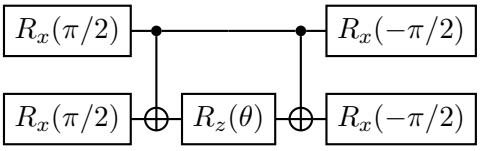



Figura 1.3: Circuito per $R_{yy}$.

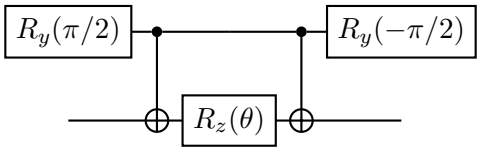



Figura 1.4: Circuito per $R_{xz}$.

### 1.1.4 Misura

Come già detto nella Sezione 1.1.1, un qubit può essere preparato in una sovrapposizione di stati, ma nel momento in cui lo si misura, si ottiene un solo autovalore, con probabilità che dipendono dai coefficienti della combinazione lineare nell'Equazione 1.1. Siccome la scelta della base su cui scrivere lo stato è arbitraria, la misura non avviene necessariamente sulla base $\{|0\rangle, |1\rangle\}$, anche detta *Z-basis*, ma su una qualsiasi altra coppia di vettori ortonormali, come $\{|+\rangle, |-\rangle\}$, la *X-basis*, perciò sorge spontaneo chiedersi come cambiano le ampiezze nel momento in cui si cambia la base di misura. Per capirlo si prenda il qubit descritto dall'Equazione 1.1, è rappresentato sulla base X, quindi si ottiene $|0\rangle$ con probabilità $|\alpha|^2$ e $|1\rangle$ con probabilità $|\beta|^2$. Ora lo si vuole misurare sulla base Y, dunque si cambia la base usando la definizione (Equazione 1.3) e si ottengono le nuove probabilità:

$$\frac{\alpha+\beta}{\sqrt{2}}|+\rangle + \frac{\alpha-\beta}{\sqrt{2}}|+\rangle$$

Si noti che, siccome la probabilità si ottiene facendo il modulo quadro dei coefficienti, le fasi globali sono ininfluenti.

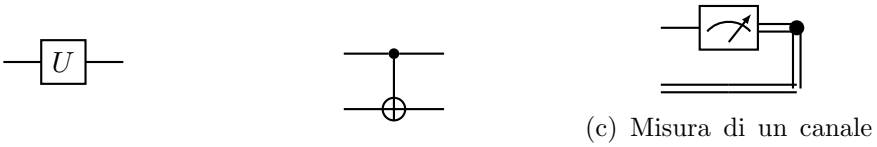
All'atto pratico quando si vuole cambiare la base di misurazione è sufficiente ruotare il qubit applicando gate opportuni; ad esempio, una misura sulla base X non è altro che la misura sulla base Z, ma dopo aver applicato il gate H. In Tabella 1.2 si trovano i gate da applicare per compiere misure nelle basi più comuni.

| Base | Gate |
|---|---|
| $Z$ | $I$ |
| $X$ | $H$ |
| $Y$ | $HS^\dagger$ |

Tabella 1.2: Gate da applicare per il cambio di base.

### 1.1.5 Circuiti

Come già visto nell sezione precedente, quando si vuole operare sui qubit torna utile rappresentare le operazioni graficamente, il formalismo che permette di farlo è quello dei circuiti. Si costruiscono diagrammi formati da




(a) Generico gate U che agisce sul singolo qubit. La notazione è usata per $X$, $H$, etc.

(b) Controlled not gate (CNOT), il qubit superiore è il controllo, quello inferiore il target.

(c) Misura di un canale quantistico, i fili disegnati con uno spessore rappresentano canali classici.

Figura 1.5: La principale notazione utilizzata del disegnare diagrammi di circuiti.

tanti *wire*, o cavi, quanti sono i qubit che si utilizzano, e vi si rappresentano i gate e le misure da applicare ai qubit. Le operazioni vengono applicate a partire da sinistra. In Figura 1.5 sono rappresentati i principali simboli utilizzati.

### 1.1.6 Qiskit

Uno dei principali strumenti utilizzati per programmare computer quantistici è *Qiskit* [24], un pacchetto Python open source sviluppato da IBM, che permette di costruire ed eseguire programmi su simulatori di quantum computer, oppure accedere in cloud ad hardware quantistico reale. Il workflow consiste fondalmentalmente nei seguenti passi:

1. **Costruzione.** Creazione di un circuito quantistico, o in più in generale un programma, che si vuole eseguire.

2. **Esecuzione.** Si esegue il codice costruito in precedenza su un backend, ovvero una macchina reale o simulata.

3. **Analisi.** Ottenuti i risultati, Qiskit ne permette anche un'accurata analisi.

In questa tesi, tutti i codici sono scritti con la versione 0.44 di Qiskit.

#### Circuiti e primitive

Di seguito vengono elencate le principali operazioni eseguibili tramite Qiskit.

- **Creazione di circuiti.** Come prima cosa è necessario creare l'oggetto `QuantumCircuit`, chiede in argomento il numero di qubit che compongono il circuito; dunque si aggiungono i gate e le eventuali misure. È possibile trovare l'elenco dei gate applicabili su [24]. Di seguito il codice che implementa circuito in Figura 1.2.

```
from qiskit import QuantumCircuit

qc = QuantumCircuit(2)
qc.cx(0,1)
qc.rz(theta, 1)
qc.cx(0,1)
```

- **Esecuzione dei circuiti.** Dato un certo circuito `qc`, è possibile scegliere una `backend`, che sia un simulatore o una macchina reale ed il numero di volte, `shots`, che il circuito è da eseguire.

```
from qiskit import execute

results=execute(qc, backend, shots=1000)
results = results.result()
```

- **Eseguire primitive.** Nonostante sia possibile utilizzare unicamente i circuiti per ogni operazione, Qiskit mette a disposizione due classi di primitive, cioè funzioni che semplificano il codice necessario per ricavare certe informazioni dai circuiti. Sono l'*estimator*, che calcola in automatico i valori di aspettazione di un osservabile,

$$\langle O \rangle = \langle \psi | O | \psi \rangle$$

e il *sampler*, che ricava la quasi-probability distribution del circuito, ovvero la frequenza con cui i vari autostati vengono misurati. Di seguito un esempio dell'utilizzo dell'estimator.

```
from qiskit.primitives import Estimator

estimator = Estimator()
results = estimator.run(qc, observable).results()
```

### Backend

Con backend, si intende la macchina, reale o meno, su cui un circuito viene eseguito. Per quanto riguarda le macchine simulate, su Qiskit ne possono essere utilizzate di vario tipo:

- **Statevector simulator.** Si tratta di un simulatore esatto, che calcola i circuiti costruendone i vettori di stato e valutando matematicamente i gate come operatori unitari. Non serve calcolare le ampiezze eseguendo più volte il circuito, in quanto vengono ottenute algebricamente.

- **Aer simulator.** Si tratta di un simulatore configurabile, che emula il comportamento dell'hardware reale a cui si possono applicare diversi modelli di rumore (cfr. Sezione 1.2.4). I circuiti possono essere eseguiti più volte (*shots*), in modo da ricostruire le probabilità con cui si misurano gli autovalori.

Per quanto riguarda le backend reali, Qiskit è in grado di interagire con diverse tecnologie di hardware, ma la maggior compatibilità si ha con i computer quantistici di IBM, a cui si può accedere tramite la piattaforma online *IBM Quantum experience* [7]. Se si sceglie una backend reale è bene tener conto che ogni hardware possiede solo certi gate, è quindi necessario tradurre il circuito perché la macchina possa eseguirlo; questo passaggio viene automaticamente eseguito da Qiskit attraverso il modulo *transpile*.

## 1.2 Quantum noise

### 1.2.1 Noisy Intermediate-Scale Quantum devices

Si supponga di eseguire un circuito come quello in Figura 1.6. È composto da due qubit, entrambi nello stato $|0\rangle$, al primo viene applicato l'Hadamard gate e, successivamente, fa da controllo per un CNOT con il secondo. Lo stato risultante è

$$\frac{|00\rangle + |11\rangle}{\sqrt{2}}$$

gli autovalori misurabili sono dunque 00 oppure 11, ed occorrono entrambi con probabilità del 50%.

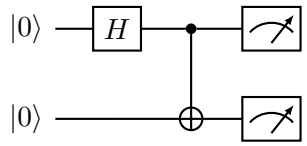



Figura 1.6: Diagramma del circuito eseguito.

Eseguendo il circuito 1000 volte, ci si aspetta di ottenere 00 e 11, entrambi circa 500 volte, con un certo errore statistico dovuto al fatto che la misura è casuale; mentre è impossibile che escano 01 o 10. Nell'istogramma in Figura 1.7, sono rappresentati i risultati dell'esecuzione del circuito su simulatore esatto e quelli ottenuti tramite hardware reale. Si nota subito che, nel

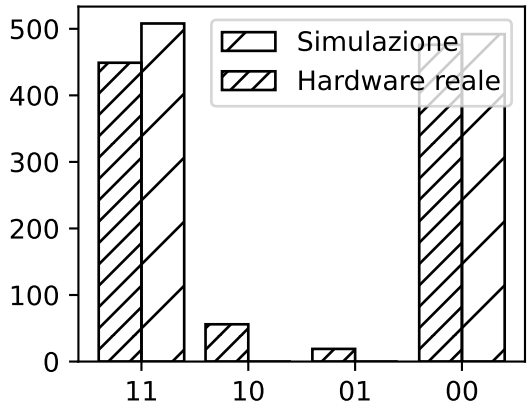



Figura 1.7: Risultati della misura del circuito con un simulatore senza rumore e sul computer quantistico "ibmq_quito".

secondo caso, si misurano valori non ammessi, significa che sono stati commessi degli errori nell'esecuzione del circuito. Si parla di *Quantum Noise*, o rumore quantistico, per riferirsi all'insieme degli errori e ai suoi effetti sulla computazione. Idealmente, si vorrebbe che la computazione avvenisse in un sistema chiuso, ovvero completamente isolato dall'ambiente circostante, ma questo subirà sempre anche le interazioni con l'esterno, in tal caso si parla di sistemi aperti. Sono le interazioni indesiderate che si traducono in rumore. Gli attuali computer quantistici sono detti *Noisy Intermediate-Scale Quantum devices* (NISQ) [12], cioè macchine il cui rate di errori commessi è ancora troppo alto perché l'esecuzione sia affidabile. Questo limita la lunghezza[4] e la complessità dei circuiti che è possibile calcolare, in quanto maggiore è la quantità di gate, tanto più alta è la probabilità di compiere errori. In particolare, non sono possibili tecniche di *error correction*, algoritmi che permettono di trovare e correggere gli errori. Alla base del loro funzionamento, infatti, c'è la codifica dell'informazione in modo ridondante [19], così, se parte del messaggio fosse distrutta, sarebbe comunque possibile recuperarla, ma a costo di un aumento della lunghezza dei circuiti. Se il rumore è troppo alto, l'introduzione di più gate incrementa la probabilità di commettere errori più di quanto l'algoritmo possa correggerla. Quando si raggiungerà la capacità tecnica di costruire hardware con un rate di errore sufficientemente basso da consentire la correzione degli errori si parlerà di macchine *fault-tolerant*.

[4] Anche detta profondità.

### 1.2.2 Error mitigation

Gli algoritmi volti al miglioramento della qualità delle esecuzioni su NISQ sono detti algoritmi di *error mitigation*. Queste tecniche non sono in grado di correggere gli errori, ma il loro costo in termini di risorse e, soprattutto, l'incremento della lunghezza del circuito necessaria alla loro implementazione è molto ridotto rispetto agli algoritmi di error correction, permettendone l'utilizzo sull'attuale hardware. Di seguito verranno mostrate due delle principali tecniche di mitigazione.

#### Zero Noise Extraplation

Un primo algoritmo di mitigazione del rumore è detto *Zero Noise Extrapolation* (ZNE), proposto in maniera indipendente in [25, 26]. Viene utilizzato per ricavare il valore di aspettazione di un'osservabile $\langle A \rangle$ calcolandola a diverse intensità di rumore ed estrapolandone il valore nel limite in cui il noise è nullo. L'algoritmo si può dividere in due passaggi:

1. **Riscalare il rumore.** L'obbiettivo è ottenere il valore di aspettazione dell'osservabile in funzione del rumore, $\langle A \rangle(\lambda)$ dove $\lambda$ è una misura del noise.

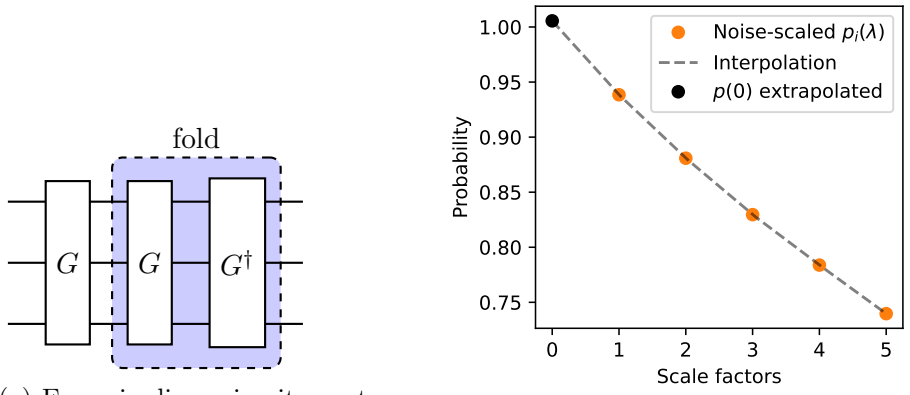
2. **Estrapolare il limite a rumore nullo.** Si cerca una certa funzione da fittare con i dati ricavati al punto precedente per estrapolare il valore di $\langle A \rangle(0)$, Figura 1.8b.

Il primo passo si può compiere in diversi modi, il più comune è detto *circuit folding* [25, 27]. Si supponga di voler eseguire un circuito $G$, per l'algebra degli operatori unitari, inserendovi i gate $GG^\dagger$, il risultato della computazione non cambia, ma ne aumenta la lunghezza e quindi il rumore viene incrementato. Ripiegando il circuito un diverso numero di volte, i.e. aggiungendo più volte l'identità $GG^\dagger$, lo si può valutare a diversi livelli di rumore (Figura 1.8a). Un'alternativa al circuit folding, ma concettualmente simile, è la *identity insertion*, che consiste nell'aggiungere un numero crescente di layer di gate che computino l'identità. Una volta ottenuto il valore dell'osservabile a diverse intensità di rumore, si può procedere all'estrapolazione al limite nullo, per farlo bisogna trovare una certa funzione $f(\lambda, \vec{\theta})$ che dipenda dai parametri $\vec{\theta}$ tale per cui

$$\langle A \rangle(\lambda) = f(\lambda, \vec{\theta})$$

Utilizzando i dati ricavati al passo precedente si possono trovare i parametri di $f$ tramite un fit ed estrapolare così il valore per $\lambda = 0$, che sarà la miglior stima del valore di aspettazione dell'osservabile in assenza di rumore. Generalmente i modelli di fit utilizzati sono funzioni polinomiali o esponenziali [13].

All'atto pratico, spesso non viene estrapolato direttamente il valore di aspettazione dell'osservabile, ma si utilizza la definizione $\langle A \rangle = \sum_i p_i a_i$,




(a) Esempio di un circuito usato per riscalare il rumore tramite folding.

(b) Estrapolazione a zero noise.

Figura 1.8: Rappresentazione dei due passi dello ZNE.

dove le $a_i$ sono gli autovalori e le $p_i$ le rispettive probabilità. Lo ZNE viene applicato alle probabilità piuttosto che direttamente a $\langle A \rangle$, Figura 1.8b. Il motivo sta nel fatto che lo ZNE è una tecnica approssimata, e ricavare le $p_i$ permette un maggiore controllo sui risultati. Ad esempio, la probabilità di ottenere uno degli autovalori potrebbe risultare maggiore di 1 o minore di 0 dall'estrapolazione. O ancora, a causa del rumore potrebbero risultare autovalori che non ci si aspetta fisicamente. Con un'analisi delle ampiezze degli autovalori è possibile rimuovere queste incongruenze.

### Probabilistic Error Cancellation

Assieme allo ZNE, la *Probabilistic Error Cancellation* o PEC, presentata in [26], è una delle principali tecniche di mitigazione degli errori. L'idea di base consiste nell'aggiungere opportuni gate al circuito che si vuole eseguire per annullare il rumore. Gli step principali in questo algoritmo sono:

1. **Trovare un modello di rumore per l'hardware utilizzato.** Tramite metodi Monte Carlo è possibile trovare i coefficienti di una combinazione di gate reali che rappresenti il rumore su un circuito.

2. **Cancellazione del rumore con i gate trovati.** Conoscendo il modello di rumore lo si può invertire di modo che cancelli gli errori.

Una qualsiasi backend reale è in grado di implementare solo un set finito di operazioni rumorose $\{O_1, \ldots, O_m\}$, perciò il singolo gate $G$ sarà semplicemente la combinazione lineare

$$G = \sum \eta_\alpha O_\alpha$$

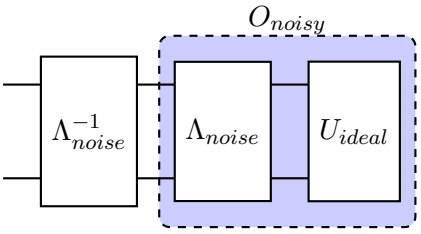



Figura 1.9: Rappresetazione schematica della PEC, gli operatori reali possono essere schematizzati come la quasi-probability distribution di opearatori reali (o viceversa), scoprendo il legame tra i due si può costruire un circuito che cancelli il rumore.

di conseguenza il valore di aspettazione di una osservabile $\langle A\rangle_{noisy}$ è

$$\langle A\rangle_{noisy} = \mathrm{tr}(AO_\alpha(|0\rangle\langle 0|^{\otimes n}))$$

Chiamando $\{U_1, \ldots, U_k\}$ un set di operazioni ideali, il valore di aspettazione noiseless $\langle A\rangle_{ideal}$ sarà

$$\langle A\rangle_{ideal} = \mathrm{tr}(AU_\beta(|0\rangle\langle 0|^{\otimes n}))$$

Se, per ogni gate ideale $\{U_\beta\}$, si trovasse una certa distribuzione di probabilità $P_\beta(\alpha)$, detta *quasi-probability distribution*, con cui scrivere $U_\beta$ come combinazione lineare di $\{O_\alpha\}$, si avrebbe

$$\langle A\rangle_{ideal} = \mathrm{tr}(AP_\beta(\alpha)\langle A\rangle_{noisy})$$

Studiando il rumore tramite metodi Monte Carlo, si può trovare la distribuzione $P_\beta(\alpha)$ per i singoli gate implementati dalla backend, quindi è possibile costruire un circuito che annulli gli effetti del rumore sulla computazione.

In altre parole, si può immaginare che il circuito da implementare sia in realtà la combinazione di un certo modello[5] di rumore $\Lambda_{noise}$ con un circuito ideale $U_{ideal}$.Trovando una corretta rappresentazione tramite le quasi-probability distributions di $\Lambda_{noise}$, lo si può invertire per annullarne (Figura 1.9).

La necessità di studiare rumore rende la PEC una tecnica piuttosto difficile da scalare e più costosa in termini di sampling, in quanto il modello $\Lambda_{noise}$ deve essere sufficientemente accurato e studiato. In [28], viene proposto un modello di rumore per circuiti formati da un arbitrario numero di gate a due qubit noto come *Pauli-Lindblad noise model*. Consiste in un prodotto di matrici di Pauli, pesate con certi coefficienti. Questi vengono ricavati eseguendo circuiti con diverse profondità, per avere vari livelli di rumore, ed eseguendo un fit per ricavare la miglior stima[6].

[5] Con modello di rumore si intende una descrizione matematica dello stesso, cfr. Sezione 1.2.4

[6] Si tratta di un concetto ben diverso da quello della ZNE, qui il fit serve per stimare dei coefficienti, non per azzerare il rumore.

#### Mitiq

Per l'implementazione delle tecniche di error mitigation, è molto popolare l'utilizzo di Mitiq [29], un pacchetto di Python compatibile con Qiskit. Si tratta di un toolkit che, alla versione 0.28, è in grado di implementare i più comuni algoritmi di mitigazione del rumore tra cui lo ZNE e la PEC.

### 1.2.3 Descrizione matematica del rumore: operatore densità

La descrizione di un sistema tramite la funzione d'onda $|\psi\rangle$ è possibile quando se ne conosce esattamente lo stato. Risulta facile trovare esempi di sistemi fisici in cui questa condizione non è verificata, come l'emissione di fotoni da una sorgente non polarizzata, oppure, come nel caso del quantum noise, l'influenza dell'ambiente su un qubit in un quantum computer. La descrizione di questi ultimi sistemi necessita di un nuovo formalismo che permetta di fare previsioni anche in caso in cui si possiedano informazioni incomplete.

Si consideri un sistema in cui una sorgente di fotoni emette luce polarizzata in una certa direzione fissata, descritta dalla funzione d'onda $|\psi_1\rangle$, con una certa probabilità $p_1$, e luce polarizzata in direzione ortogonale alla precedente, descritta da $|\psi_2\rangle$, con una certa probabilità $p_2$. Tale situazione può essere facilmente estesa ad un numero $n$ di funzioni d'onda $|\psi_k\rangle$ con le rispettive probabilità $p_k$. I singoli vettori $|\psi_k\rangle$ sono detti *stati puri*, o *pure state*, mentre il sistema totale è una *miscela statistica* di stati, o *mixed state*.

È bene notare la differenza tra una miscela di stati e lo stato puro descritto dalla sovrapposizione di funzioni d'onda

$$|\psi\rangle = \sum_i c_i |\psi_i\rangle \tag{1.9}$$

In quest'ultimo caso la probabilità di trovare il sistema nell'autostato $|\psi_i\rangle$ è descritta da $|c_i|^2$, il che non coincide con un sistema di stati miscelati in cui la probabilità di essere nello stato $|\psi_i\rangle$ è $p_i = |c_i|^2$, infatti così facendo non si considererebbero i fenomeni di interferenza dovuti alla sovrapposizione delle funzioni d'onda dell'Equazione 1.9. La conseguenza di questa osservazione è che è impossibile scrivere un mixed state in una formula analoga a quella dell'Equazione 1.9.

Per studiare miscele di stati si introduce l'*operatore densità*, a cui è associata una *matrice densità* [19, 30, 31].

#### Stati puri

Come primo approccio si mostrerà la descrizione tramite operatore densità dello stato puro

$$|\psi\rangle = \sum_k c_k |\varphi_k\rangle$$

Si consideri una certa osservabile $\hat{A}$, il suo valore di aspettazione su $|\psi\rangle$ vale

$$\langle \hat{A} \rangle = \langle \psi | \hat{A} | \psi \rangle = \sum_{m,n} c_m^* c_n A_{mn} \tag{1.10}$$

dove $c_m^* c_n$ sono gli elementi della matrice associata al proiettore sullo stato puro. Si definisce l'operatore densità come

$$\rho \equiv |\psi\rangle \langle \psi| \tag{1.11}$$

gli elementi della matrice densità saranno, sulla base $\{\varphi_k\}$, $\rho_{mn} = c_n^* c_m$. Utilizzando questo nuovo operatore è possibile esprimere la media dell'osservabile $\hat{A}$, a partire dall'Equazione 1.10, nel seguente modo:

$$\begin{aligned} \langle \hat{A} \rangle &= \sum_{m,n} \rho_{mn} A_{m,n} \\ &= \sum_{m,n} \langle \varphi_n | \rho | \varphi_m \rangle \langle \varphi_m | \hat{A} | \varphi_n \rangle \\ &= \mathrm{tr}(\rho \hat{A}) \end{aligned}$$

Dove si passa dalla seconda alla terza riga usando la relazione di completezza $\sum_m |\phi_m\rangle \langle \phi_m| = I$ e la definizione di traccia, cioè $\mathrm{tr}(\hat{O}) = \sum_i \hat{O}_{ii}$.

Infine, è possibile scrivere in termini di $\rho$ anche la probabilità $P(a_n)$ di misurare un autovalore di $\hat{A}$:

$$P(a_n) = \langle \psi | P_n | \psi \rangle = \mathrm{tr}(\rho P_n) \tag{1.12}$$

Come si vedrà nel prossimo paragrafo la vera utilità della descrizione tramite operatore densità è nella manipolazione di stati miscelati.

### Mixed states

Il passo successivo è quello di estendere il formalismo in questione anche agli stati miscelati. Si prenda l'insieme di stati puri[7] $\{p_k, |\psi_k\rangle\}$, si vuole trovare la probabilità $P(a_n)$ di misurare un certo autostato $a_n$ dell'operatore introdotto nella sezione precedente; detto $P_n$ il proiettore sugli autostati di $\hat{A}$, si ha:

$$P_k(a_n) = \langle \psi_k | P_n | \psi_k \rangle$$

Considerato che ogni stato puro partecipa allo stato miscelato con un certo peso $p_k$, la probabilità di misurare $a_n$ sul sistema è

$$P(a_n) = \sum_k p_k P_k(a_n)$$

[7] Spesso noto in lingua inglese come *ensemble of pure state.*

in analogia con l'Equazione 1.12, detto $\rho_k$ l'operatore densità dello stato puro $|\psi_k\rangle$, si ha

$$P(a_n) = \sum_k p_k \mathrm{tr}(\rho_k P_n) = \mathrm{tr}(\rho P_n)$$

definendo l'operatore densità per stati miscelati:

$$\rho \equiv \sum_k p_k \rho_k = \sum_i |\psi_i\rangle \langle\psi_i|$$

Si dice che il set $|\psi_i\rangle$ genera $\rho$. Discende infine da tale definizione che

$$\langle \hat{A} \rangle = \mathrm{tr}(\rho A)$$

**Proprietà e descrizione di sottosistemi**

L'operatore densità è caratterizzato da due importanti proprietà. La prima, che deriva dalla definizione di probabilità[8] è

$$\mathrm{tr}(\rho) = \sum_k p_k = 1$$

La seconda proprietà è uno strumento che permettere di distinguere operatori densità che si riferiscono a stati puri da quelli che si riferiscono a una miscela. Considerando il quadrato dell'operatore, si ha

$$\mathrm{tr}(\rho^2) \leq 1$$

dove l'uguaglianza è valida se e solo se $\rho$ si riferisce a uno stato puro; anche tale proprietà è una conseguenza della definizione di probabilità[9].

Si può notare che l'insieme $\{p_i, |\psi_i\rangle\}$ genera un unico operatore densità, ma non è vero il contrario, si dimostra [19] che due set $\{p_i, |\psi_i\rangle\}$ e $\{p_i, |\varphi_i\rangle\}$, generano la stessa matrice densità se e solo se

$$|\psi_i\rangle = \sum_k u_{ik} |\varphi_k\rangle$$

dove gli $u_{ij}$ sono elementi di una matrice complessa unitaria.

Si supponga ora di avere due sistemi fisici A e B che formano un sistema globale A+B; si vorrebbe trovare un modo per descrivere i sottosistemi a partire dall'operatore densità di A+B, $\rho^{AB} = \rho^A \otimes \rho^B$. Serve introdurre la nozione di *traccia parziale*, definita da

$$\mathrm{tr}_A(\rho^{AB}) \equiv \sum_k \langle b_k | \rho^{AB} | b_k \rangle$$

dove i $|b_k\rangle$ sono gli elementi della base in cui è descritto il sistema $B$. Dalla definizione si ottiene che

$$\rho^B = \mathrm{tr}_A(\rho^{AB})$$

[8] In particolare, come già notato, la somma di tutte le probabilità $p_k$ di un ensemble statistico deve essere 1.

[9] In particolare, $p_k \leq 1$, perciò $p_k^2 \leq 1$.

### Quantum operations

Fino ad ora si è visto come descrivere e fare predizioni sulla misura di sistemi quantistici tramite il formalismo dell'operatore densità. Per completare la trattazione è necessario introdurre un concetto matematico che permetta di ricostruire l'evoluzione di un certo stato, tale nozione è nota come *quantum operation* [19], ed è la mappa $\mathcal{E}$ tale che

$$\rho' = \mathcal{E}(\rho)$$

Se, per esempio, lo stato puro $|\psi\rangle$, a cui è associata la matrice densità $\rho$, si evolve tramite l'operatore unitario $\hat{U}$, allora la quantum operation associata a tale evoluzione è[10]

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

Per scopo di questa tesi, quanto si vuole fare è trovare un modo per esprimere l'interazione tra il sistema di interesse (i qubit) e l'ambiente circostante. Esistono diverse strategie per raggiungere tale obbiettivo, qui verrà utilizzata quella nota come *quantum-sum representation.* Si consideri un certo sistema principale $\rho$, e si indichi con $\rho_{env}$ la matrice densità dell'ambiente. Qualora si voglia compiere una certa operazione $\hat{U}$ sul sistema aperto, che, per semplicità, si assume sia $\rho \otimes \rho_{env}$, il sistema finale può essere descritto tramite l'operazione $\mathcal{E}(\rho)$. Usando le tracce parziali, detta $\{e_i\}$ una base di $\rho_{env}$, che si suppone essere nello stato iniziale $\rho_{env} = |e_0\rangle\langle e_0|$, si può scrivere

$$\mathcal{E}(\rho) = \sum_i \langle e_k|\, \hat{U}\big[\rho \otimes |e_0\rangle\langle e_0|\,\big]\hat{U}^\dagger\, |e_k\rangle$$

dove è possibile definire gli *operation elements* (o *operatori di Kraus*) di $\mathcal{E}(\rho)$, $E_k \equiv \langle e_k|\hat{U}|e_0\rangle$ e ottenere

$$\mathcal{E}(\rho) = \sum_k E_k \rho E_k^\dagger \tag{1.13}$$

Si noti che gli elementi $E_k$ agiscono sul sistema principale; significa che ne caratterizzano la dinamica senza dover introdurre esplicitamente l'ambiente, che spesso è ignoto.

## 1.2.4 Modelli di rumore

Per poter studiare il quantum noise è utile avere a disposizione degli oggetti matematici con cui descriverlo, questi oggetti sono detti *modelli di rumore.* La creazione di un modello permette di avere simulatori di quantum computer che replichino il comportamento di all'hardware reale, oppure che isolino specifici tipi di errori e ne varino il rate. Per farlo si utilizza il formalismo delle matrici densità e delle quantum operations.

[10] Si deriva da $|\psi'\rangle\langle\psi'| = \hat{U}\,|\psi\rangle\langle\psi|\,\hat{U}^\dagger$, con $\rho = |\psi\rangle\langle\psi|$ e $\rho' = |\psi'\rangle\langle\psi'|$

Di seguito verranno elencati i principali tipi di errore che possono avvenire durante la computazione su un quantum computer. Un modello di rumore realistico, sarà una combinazione di tutti questi canali di errore con le opportune probabilità.

### Bit e phase flips

L'operazione più semplice da modellare è il *bit flip*, descrive il caso in cui un qubit ha una certa probabilità $p$ di scambiare le ampiezze degli elementi della base $|0\rangle$ e $|1\rangle$ [19, 20]; il gate quantistico che compie questo scambio è il NOT, o X, perciò gli operatori di Kraus associati al bit flip sono:

$$E_0 = \sqrt{1-p}I \qquad E_1 = \sqrt{p}X$$

Il primo operatore descrive il caso in cui il qubit rimane nello stato iniziale, mentre il secondo il caso in cui avviene il bit flip. L'operazione quantistica su uno stato $\rho$ corrispondente è

$$\mathcal{E}(\rho) = E_0 \rho E_0^\dagger + E_1 \rho E_1^\dagger$$

ottenuta mediante l'applicazione dell'Equazione 1.13.

Concettualmente uguale al bit flip è il *phase flip*, che descrive il caso in cui a scambiarsi le ampiezze siano gli elementi della base $|+\rangle$ e $|-\rangle$ con probabilità $p$; il gate che scambia la fase è $Z$, perciò

$$E_0 = \sqrt{1-p}I \qquad E_1 = \sqrt{p}Z$$

Generalmente i due flip vengono descritti insieme tramite l'operazione *bit-phase flip*; si osservi che la combinazione dei gate che descrivono un simultaneo scambio di fase e di ampiezza è $iXZ = Y$, perciò gli elementi dell'operazione sono

$$E_0 = \sqrt{1-p}I \qquad E_1 = \sqrt{p}Y$$

Nell'hardware attuale, la frequenza con cui avviene questo tipo di errore è di circa $10^{-3}$.

### Canale di depolarizzazione

Il *canale di depolarizzazione* è l'operazione che descrive un qubit che ha una certa probabilità $p$ di essere depolarizzato, cioè di finire in uno stato massimamente miscelato [19]. Si tratta dello stato la cui matrice densità è proporzionale all'identità; nel caso del singolo qubit si ha

$$\frac{1}{2}(|0\rangle\langle 0| + |1\rangle\langle 0|) = \frac{I}{2}$$

con $I$ la matrice identità, definizione che può essere intuitivamente estesa a uno stato con $n$ qubit osservando che l'unico stato proporzionale a $I$ è $I/n$. L'operazione quantistica di depolarizzazione del singolo qubit è

$$\mathcal{E}(\rho) = p\frac{I}{2} + (1-p)\rho$$

### Amplitude e phase damping

Una delle possibili cause di errore in un computer quantistico è la perdita di energia del sistema. Non essendo perfettamente isolati dall'ambiente i quantum computer, interagiscono con esso: possono essere colpiti da raggi cosmici, oppure aumentare la loro temperatura emettendo un fotone, oppure perdere energia per altre cause. La descrizione dei singoli processi sarebbe troppo complessa, ma tutti hanno lo stesso effetto sulla funzione d'onda, ovvero lo smorzamento dell'ampiezza. Questo errore, che prende il nome di *amplitude damping* [19], è descritto dalle seguenti matrici di Kraus

$$E_0 = \begin{bmatrix} 1 & 0 \\ 0 & \sqrt{1-\gamma} \end{bmatrix} \qquad E_1 = \begin{bmatrix} 0 & \sqrt{\gamma} \\ 0 & 0 \end{bmatrix}$$

dove $\gamma$ dipende dal sistema e non sarà costante del tempo [32, 33].

Accanto alla dissipazione di energia esiste un fenomeno fisico tipico solo dei sistemi quantistici per cui si perde informazione senza una perdita di energia. Si tratta di un cambiamento di fase della funzione d'onda. L'operazione quantistica associata è il *phase damping* [19] ed è descritta dalle matrici

$$E_0 = \begin{bmatrix} 1 & 0 \\ 0 & \sqrt{1-\delta} \end{bmatrix} \qquad E_1 = \begin{bmatrix} 0 & 0 \\ 0 & \sqrt{\delta} \end{bmatrix}$$

anche in questo caso $\delta$ dipende dal sistema ed è funzione del tempo [32, 33].

Per ricavare i parametri da cui dipendono i damping, si utilizzano i *tempi di coerenza* [32], ovvero il tempo durante il quale l'ampiezza dell'onda si può assumere costante, sono indicati con $T_1$ e $T_2$, il primo, detto *relaxation time* ed è il tempo impiegato al qubit per passare da $|1\rangle$ a $|0\rangle$, il secondo, detto *dephasing* time, per passare da $|+\rangle$ a $1/2\,[|+\rangle + |-\rangle]$. Per i computer quantistici di IBM [7], i tempi di coerenza sono nell'ordine dei $150\mu s$ per $T_1$ e dei $50\mu s$ per $T_2$. Il legame tra le matrici di Kraus e questi tempi è [32]

$$\gamma = 1 - e^{-\tau/T_1} \qquad \delta = 1 - e^{-2\tau/T_\varphi}$$

dove $\tau$ è il tempo necessario a compiere l'operazione, mentre $T_\varphi$ è definita da $T_\varphi = 2T_1T_2/(2T_1 - T_2)$.

### Errori di misura

Infine, uno degli errori che più limitano l'aumento dei qubit nei computer attuali è il *Readout error*, ovvero l'errore nella misura del circuito [34]. La

probabilità con cui avvengono errori di questo tipo è, sempre per i computer di IBM [7], nell'ordine di $10^{-2}$, piuttosto alta se confrontata con le probabilità precedenti. Si tratta di una tipologia di errore ben diversa da quelle incontrate fino ad ora, infatti gli errori non si accumulano con la lunghezza del circuito in quanto avvengono solo a fine ciclo [35]. Il modello per il readout error è una matrice, detta *confusion matrix*, $A \in M_{2^n \times 2^n}(\mathbb{R})$, dove $n$ è il numero di qubit, e l'entrata $A_{jk}$ è la probabilità di misurare $j$ se lo stato preparato realmente è $k$, per due qubit si ha, ad esempio,

$$A = \begin{pmatrix} P(00|00) & P(01|00) & P(10|00) & P(11|00) \\ P(00|01) & P(01|01) & P(10|01) & P(11|01) \\ P(00|10) & P(01|10) & P(10|10) & P(11|10) \\ P(00|11) & P(01|11) & P(10|11) & P(11|11) \end{pmatrix}$$

Questo genere di errore fa parte di una categoria più ampia detta SPAM, o *State Preparation and Measurement errors* [36], che comprende anche gli errori di preparazione dello stato iniziale, ma l'errore dominante è di gran lunga quello sulla misura.

### Implementazione dei modelli di rumore tramite Qiskit

Qiskit permette di costruire dei modelli di rumore artificiali, per uno studio più approfondito degli algoritmi. Di seguito un esempio guidato su come costruire un basilare modello.

Le funzioni necessarie a questo scopo sono contenute nel modulo di *qiskit_aer* chiamato *noise*.

```
from qiskit import QuantumCircuit
from qiskit_aer.noise import NoiseModel,
                             QuantumError,
                             pauli_error
                             amplitude_damping_error,
                             phase_damping_error
```

Il primo passo nella costruzione del rumore consiste nella definizione dei singoli errori con gli opportuni parametri.

```
bit_flip = pauli_error([('X', p_error), ('I', 1 - p_error)])
phase = phase_damping_error(param_phase)
amplitude = amplitude_damping_error(param_amp)
```

Infine bisogna applicare tutti gli errori definiti ai qubit di interesse, scegliendo opportunamente quali siano i gate rumorosi. Nel seguente esempio, i

gate rumorosi sono quelli che implementano le rotazioni, e tutti i qubit sono affetti dal rumore.

```
phase_amplitude = phase.compose(amplitude)
errors = bit_flip.compose(phase_amplitude)
noise_model.add_all_qubit_quantum_error(errors,["rz","rx","ry"])
```

## 1.3 Variational Quantum Algorithms

Per quanto detto, sui dispositivi NISQ la lunghezza dei circuiti e il numero di qubit sono seriamente limitati, rendendo inapplicabili su larga scala tanti degli algoritmi che porterebbero al vantaggio di questo paradigma di computazione su quello classico. È con l'obiettivo di rendere competitivi anche i computer attuali che nascono i *Variational Quantum Algorithms* [14, 15], o VQA, che usano computer quantistici solo per la valutazione di circuiti parametrizzati, lasciando l'ottimizzazione dei parametri a computer classici.

### 1.3.1 Struttura di un VQA

Il funzionamento di un algoritmo variazionale quantistico si può schematizzare nei passaggi seguenti, rappresentati anche in Figura 1.10.

1. **Definizione di una cost function.** Si tratta di una funzione il cui minimo corrisponde alla soluzione del problema, dipende da un certo vettore di parametri.

2. **Scelta di un ansatz.** Si tratta di un circuito, i.e. un operatore, funzione degli stessi parametri della cost function.

3. **Ottimizzazione classica dei parametri.** Si cercano i parametri che minimizzano la cost function usando informazioni ricavate dalla computazione quantistica, spesso un gradiente, e ottimizzatori classici.

Alla base dei VQA c'è il fatto che lavorare su un circuito parametrizzato che approssimi uno stato reale è meno costoso che lavorare direttamente su questo.

#### Cost function

Una *cost function* $L(\theta)$, è una funzione che dipende da $p$ parametri $\theta$. Contiene tutte le informazioni necessarie alla soluzione del problema a cui il VQA è applicato, nel senso che al suo minimo globale in $\theta$, deve corrispondere la sua soluzione. Siccome gli algoritmi variazionali sono pensati all'applicazione su dispositivi NISQ, deve richiedere un numero limitato di qubit, così

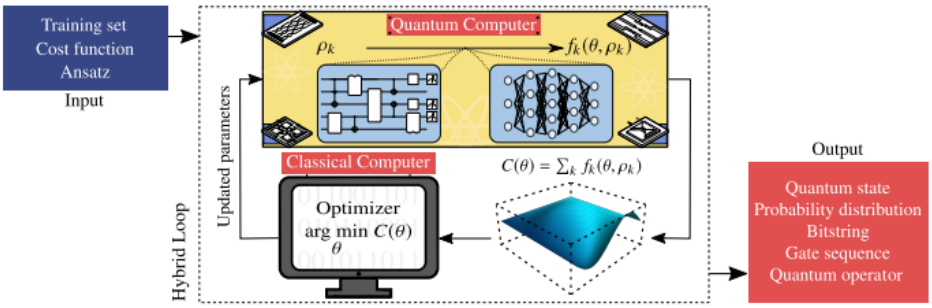



Figura 1.10: Diagramma, tratto da [14], del funzionamento dei VQA.

come limitata deve essere la lunghezza del circuito. Un esempio piuttosto comune di cost function, è l'energia di un sistema con Hamiltoniana $H$.

### Ansatz

L'ansatz è un circuito che dipende dai parametri $\theta$, sceglierlo è un passaggio molto delicato in un VQA, in quanto dalla sua struttura dipendono i parametri e il funzionamento dell'algoritmo; deve, infatti, approssimare bene lo stato esatto che si avrebbe in un algoritmo non variazionale. In generale, lo si può indicare come un operatore unitario $U(\theta)$,costituito dal prodotto di un certo numero di rotazioni, applicato ad uno stato iniziale:

$$U(\theta) = R_n \dots R_1 |0\rangle$$

### Gradiente

Nonostante si sia parlato di ottimizzazione classica, questa in realtà coinvolge anche certe informazioni ricavate con un computer quantistico. In particolare, il calcolo della cost function, e quindi del suo gradiente, avviene su questo tipo di hardware, in quanto la sua valutazione classica richiede risorse esponenziali nella taglia del circuito. Trovare il gradiente è un'operazione alquanto complicata e richiede spesso l'utilizzo di teorie perturbative o altri tipi di approssimazione. Fortunatamente, nel caso in cui l'ansatz sia un prodotto tensoriale di matrici di Pauli, è possibile utilizzare una regola nota come *parameter-shift rule* [37, 38, 14, 15], per cui la $j-$esima componente del gradiente è calcolata tramite

$$g_j = \frac{\partial L}{\partial \theta_j} = \frac{L(\theta + s e_j) - L(\theta + s e_j)}{2 \sin s} \tag{1.14}$$

con $s$ reale[11] e $e_i$ un versore nullo eccetto che nella $j-$esima entrata. Trovare il gradiente richiede la misura della cost function due volte per ogni elemento di $\theta$.

[11] La scelta dipende dall'hardware.

### Ottimizzazione

L'ultimo step nei VQA è l'ottimizzazione dei parametri, questa avviene spesso, seppur esistano altri metodi, utilizzando il gradiente valutato tramite il quantum computer. In questa sede ci si occuperà di solo di algoritmi che lo usano. L'idea alla base è sempre la stessa, ovvero iterare un certo insieme di operazioni cambiando opportunamente i parametri fino a che il valore della cost function $L(\theta)$ non scende sotto una certa soglia $L_{min}$. Di seguito tre famosi algoritmi di ottimizzazione derivanti dal machine learning.

***Gradient descent*** Spesso indicato con la sigla (GD) è un algoritmo di ottimizzazione che sfrutta la proprietà del gradiente di essere ortogonale alle superfici equipotenziali, cioè a crescita nulla, in pratica il gradiente è un vettore che indica in quale direzione la crescita della funzione è maggiore. Per minimizzare i parametri si procede quindi in direzione opposta. L'$n-$esimo step di ottimizzazione dello GD avviene come segue

$$\theta_n = \theta_{n-1} - \eta g_{n-1}$$

dove $g_{n-1} = \nabla_\theta L(\theta_{n-1})$. L'ottimizzazione si conclude quando $L(\theta_n)$ ha raggiunto il valore di soglia $L_{min}$. La costante $\eta$, reale e strettamente positiva, è detta *learning rate*, si tratta di un fattore empirico che pesa il gradiente. Se $\eta$ è troppo piccolo, ogni passo nella direzione del minimo della cost function è piccolo, da cui un numero di step molto grandi per raggiungerlo; viceversa, se è troppo grande l'ottimizzazione potrebbe esserne cieca.

**Algoritmo 1** Pseudocodice per l'implementazione del gradient descent.

**Require:** $\eta \in \mathbb{R}^+$.
**Require:** $\theta_0 \in \mathbb{R}^p$ ▷ Si definiscono $p$ parametri iniziali.
$n \leftarrow 0$ ▷ Inizializzazione step di ottimizzazione.
**while** $L(\theta_n) > L_{min}$ **do**
    $n \leftarrow n + 1$
    $\theta_n \leftarrow \theta_{n-1} - \eta g_{n-1}$ ▷ Aggiornamento dei parametri.
**end while**
**return** $\theta_n$

***ADAM*** L'*adaptive moment estimation* [39], o *ADAM*, è un ottimizzatore simile al GD, ma nel quale ogni step di ottimizzazione tiene conto dei passi precedenti riscalando il gradiente di conseguenza.

L'algoritmo ottimizza la media $m_n$ e la varianza $v_n$ del gradiente al $n-$esimo step di ottimizzazione (primo e secondo momento). Sia $g_n = \nabla_\theta L(d\theta_n, \delta t)$ il gradiente calcolato al $n-$esimo step, si aggiornano i vettori appena definiti calcolando

$$m_n = \beta_1 m_{n-1} + (1 - \beta_1) g_n \qquad v_n = \beta_2 v_{n-1} + (1 - \beta_2) g_n^2$$

dove $\beta_1$ e $\beta_2$ sono parametri che regolano il decadimento esponenziale dei due momenti. Così facendo, si ottengono valori con un bias, per rimuoverlo basta dividere $m_t$ e $v_t$ per $(1-\beta_i^t)$. Infine, si aggiornano i parametri tramite

$$\theta_n = \theta_{n-1} - \alpha \frac{m_n}{\sqrt{v_n} + \varepsilon}$$

La costante $\alpha$ è detta stepsize e, accanto a $\beta_1$, $\beta_2$ e $\epsilon$, sono valori da cui dipende il corretto funzionamento di ADAM, vanno scelti in modo empirico e dipendono dal problema in questione. Per problemi di machine learning e, come si vedrà nella Sezione 3.2, anche nel caso del pVQD, i migliori valori sono: $\alpha = 0.001$, $\epsilon = 10^{-8}$, $\beta_1 = 0.9$ e $\beta_2 = 0.999$.

Come detto ADAM si basa sul gradiente, perciò il numero di misure per step di ottimizzazione è, come per GD, quello necessario alla valutazione del gradiente.

**Algoritmo 2** Pseudocodice per l'algoritmo di ottimizzazione ADAM.

**Require:** $\alpha \in \mathbb{R}$
**Require:** $\beta_1, \beta_2 \in [0, 1)$
**Require:** $\theta_0 \in \mathbb{R}^p$
$m_0 \leftarrow 0$
$v_0 \leftarrow 0$
$n \leftarrow 0$
**while** $L(\theta) > L_{min}$ **do**
    $n \leftarrow n + 1$
    $g_n \leftarrow \nabla_\theta L(\theta_{n-1})$
    $m_n \leftarrow \beta_1 m_{n-1} + (1 - \beta_1) g_n$
    $m_n \leftarrow m_n / (1 - \beta_1^n)$
    $v_n \leftarrow \beta_1 v_{n-1} + (1 - \beta_1) g_n^2$
    $v_n \leftarrow v_n / (1 - \beta_2^n)$
    $\theta_n \leftarrow \theta_{n-1} - \alpha \frac{m_n}{\sqrt{v_n} + \epsilon}$
**end while**
**return** $\theta_n$

***SPSA*** Un altro algoritmo di ottimizzazione spesso usato è il *simultaneous perturbation stochastic approximation* (SPSA) [40], come nei due casi precedenti anche qui l'ottimizzazione avviene nella forma

$$\theta_n = \theta_{n-1} - a_{n-1} g_{n-1} \tag{1.15}$$

con una sostanziale differenza nel calcolo del gradiente. Se prima, nella parameter-shift rule, ogni entrata del gradiente veniva singolarmente perturbata di una costate $s$ (Equazione 1.14), richiedendo quindi $2p$ misure della cost function, ora viene generato casualmente[12] un vettore di lunghezza $p$,

[12] Tipicamente secondo distribuzione binomiale.

$\Delta_n$, all'$n-$esimo step di ottimizzazione, da applicare ai parametri in modo da perturbali contemporaneamente. Di conseguenza, il calcolo del gradiente ha bisogno di sole 2 misure, indipendentemente dal numero di parametri. La singola componente del gradiente all'$n-$esimo step di ottimizzazione, ha equazione

$$g_{n,j} = \frac{\partial L}{\partial \theta_j} = \frac{L(\theta_n + c_n\Delta_n) - L(\theta_n - c_n\Delta_n)}{2c_n\Delta_{n,j}}$$

si noti che la differenza al numeratore, l'unica da valutare con un computer quantistico, non dipende dall'indice $j$, è quindi uguale a tutte le componenti.

I parametri $c_n$ e $a_n$ (stepsize del gradiente), sono successioni che dipendono da $n$ definite come

$$a_n = \frac{a}{(A+n)^\alpha} \qquad c_n = \frac{c}{n^\gamma}$$

con $a$, $A$, $\alpha$, $c$ e $\gamma$ costanti scelte empiricamente.

Con i valori delle costanti scelti, l'aggiornamento dei parametri avviene secondo l'Equazione 1.15. Lo schema del funzionamento di SPSA è nel diagramma seguente.

**Algoritmo 3** Pseudocodice per l'implementazione di SPSA.

**Require:** $a$, $A$, $\alpha$, $c$ e $\gamma$

$n \leftarrow 0$ ▷ Inizializzazione step di ottimizzazione.
**while** $L(\theta) > L_{min}$ **do**
  $n \leftarrow n + 1$
  $a_n \leftarrow \frac{a}{(A+n)^\alpha}$
  $c_n = \frac{c}{n^\gamma}$
  Genera casualmente $\Delta_n \in \mathbb{R}^p$ ▷ Generazione della perturbazione.
  $g_n \leftarrow \frac{L(\theta_n+c_n\Delta_n)-L(\theta_n-c_n\Delta_n)}{2c_n} \begin{pmatrix} \Delta_{n,1}^{-1} \\ \vdots \\ \Delta_{n,p1}^{-1} \end{pmatrix}$ ▷ Calcolo del gradiente.
  $\theta_n \leftarrow \theta_{n-1} - a_n g_n$ ▷ Aggiornamento dei parametri.
**end while**
**return** $\theta_n$

# Capitolo 2

# Simulation of quantum dynamics

Nel seguente capitolo si affronta la teoria della simulazione dei sistemi dinamici. Partendo (Sezione 2.1) dalla descrizione dei simulatori quantistici, si passa (Sezione 2.2) a soluzioni adatte a computer NISQ, i variational quantum simulator. Si conclude (Sezione 2.3), con la descrizione dettagliata di un algoritmo variazionale per l'evoluzione dinamica di sistemi quantistici noto come pVQD.

## 2.1 Universal quantum simulation

Nella ricerca moderna, la simulazione dei sistemi fisici è una tecnica largamente utilizzata. Con questo termine si intende la riproduzione delle proprietà di un sistema fisico e della sua evoluzione dinamica [6]. Questa si ottiene attraverso la modellizzazione matematica del problema e la risoluzione delle equazioni risultanti. Nella maggior parte dei sistemi quantistici il tempo e la memoria necessari alla simulazione scala esponenzialmente con la dimensione del sistema, rendendo inefficiente o comunque approssimata la predizione delle variabili fisiche di interesse. Da qui, l'idea di Richard P. Feynman di usare i sistemi quantistici come simulatori [2]: nasce il campo della *Quantum Simulation*.

L'obiettivo finale della ricerca nel campo della simulazione quantistica, è quello di realizzare un *Universal Quantum Simulator*, ovvero un simulatore che sia in grado di trovare l'evoluzione temporale di un qualsiasi modello fisico.

### 2.1.1 Struttura di un simulatore

Riprodurre l'evoluzione dinamica di un sistema fisico significa risolvere un'equazione differenziale, che, nel caso dei sistemi quantistici, è l'equazione di

Schrödinger

$$i\frac{\partial}{\partial t}|\psi\rangle = H|\psi\rangle$$

dove $\hbar = 1$. Si trova che [30] la trasformazione dallo stato iniziale, a uno stato evoluto è lineare, perciò esiste l'operatore unitario, detto *operatore di evoluzione temporale*,

$$U = e^{-iHt} \tag{2.1}$$

tale per cui

$$|\psi(t)\rangle = U(t)|\psi(t_0)\rangle$$

L'evoluzione del sistema è la soluzione a questa equazione. A prima vista sembra non ci siano problemi, infatti è un'equazione perfettamente risolvibile da un computer classico; si noti però che in un sistema di taglia $n$, come potrebbe essere l'interazione tra lo spin di $n$ elettroni, le matrici in gioco hanno dimensione $2^n \times 2^n$, il che rende esponenzialmente grande il set di equazioni da risolvere. Ammesso di avere un sistema localmente interagente, cioè la cui Hamiltoniana è nella forma

$$H = \sum_l H_l$$

con $H_l$ Hamiltoniane locali[1], i computer quantistici si sono dimostrati in grado [41] di replicare efficientemente l'operatore di evoluzione temporale necessario alla simulazione.

Schematicamente il funzionamento di un Universal Quantum Simulator può essere riassunto nei seguenti passi, rappresentati anche in Figura 2.1:

1. **Definizione di un modello fisico.** Consiste nel definire l'Hamiltoniana che descrive il sistema di interesse e che sia somma di Hamiltoniane locali.

2. **Rappresentazione dell'Hamiltoniana tramite matrici di Pauli.** Serve trovare una mappa che permetta di scrivere l'Hamiltoniana come una combinazione lineare di matrici di Pauli, non esiste un modo univoco per determinarla, ma si dimostra che è possibile per gran parte dei sistemi.

3. **Traduzione dell'operatore di evoluzione temporale in circuito.** L'operatore di evoluzione temporale è scritto in forma esponenziale, per tradurlo come un prodotto di termini localmente interagenti bisogna approssimare l'espansione tramite una formula nota come Suzuki-Trotter decomposition, approfondita di seguito.

[1] Che agiscono solo su un sottosistema finito.

4. **Evoluzione si uno stato iniziale.** Per compiere l'effettiva evoluzione temporale, bisogna applicarle i gate trovati al punto precedente ad uno stato iniziale preparato tramite il quantum computer (spesso $|0\rangle$).

5. **Misurare le osservabili di interesse.** Evoluto lo stato si possono trovare i valori di aspettazione delle osservabili di interesse misurando più volte il circuito.

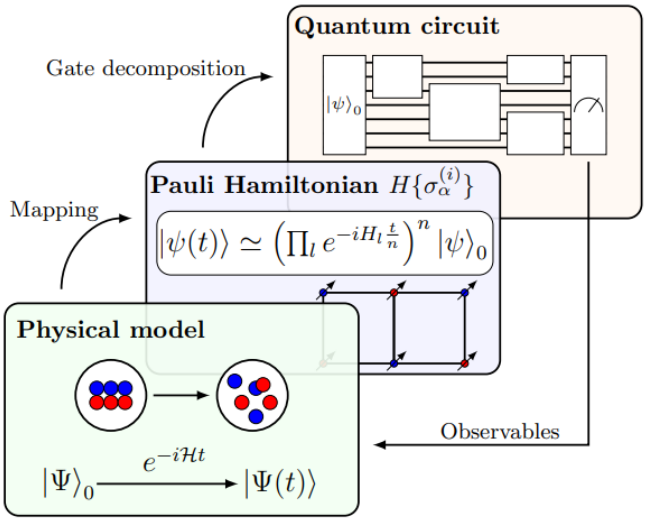



Figura 2.1: Schema del funzionamento di un Universal Quantum Simulator, figura tratta da [6].

### Decomposizione di Suzuki-Trotter

Per avere un simulatore che sia efficiente, bisogna tradurre l'operatore di evoluzione temporale in operatori locali. Si consideri un sistema dotato di un'Hamiltoniana $H$ che sia somma di Hamiltoniane locali $H_l$, in tal caso si ha

$$U(t) = e^{iHt} = e^{i\sum_l H_l t}$$

Per separare gli esponenziali bisogna ricorrere ad una formula nota come *decomposizione di Suzuki-Trotter*, o *trotterization*,

$$e^{i\sum_l H_l t} = \lim_{n\to\infty} \left(\prod_l e^{-iH_l t/n}\right)^n$$

se tutti gli operatori $H_l$ commutano allora l'identità vale già per $n = 1$. Ovviamente non è possibile implementare un limite infinito, ma si dimostra[2]

[2]Si trova la dimostrazione su [19].

che per ogni $n$ vale

$$U(t) = \left( \prod_l e^{-iH_l t/n} \right)^n + O\left(\frac{t^2}{n}\right) \tag{2.2}$$

dunque è possibile approssimare l'espansione fino alla precisione desiderata ripetendo $n-$volte i gate che implementano il termine tra parentesi, ogni ripetizione è detta *Trotter step*. In altre parole si raggiunge il tempo di evoluzione desiderato procedendo per piccoli passi temporali $t/n$.

Siccome l'Hamiltoniana è stata scritta come combinazione di operatori di Pauli, ogni termine della decomposizione di Trotter sarà nella forma $e^{-iJ\sigma_\alpha t}$ o $e^{-iJ\sigma_\alpha\sigma_\beta t}$. Il primo corrisponde a una rotazione di un angolo $Jt$ lungo l'asse $\alpha$, i.e. $R_\alpha(Jt)$. Il secondo è la rotazione di due qubit $R_{\alpha\beta(Jt)}$. Sulla traduzione in gate di queste trasformazioni si è già discusso nella Sezione 1.1.3.

### 2.1.2 Esempio: evoluzione dinamica tramite Trotter

Per esplorare meglio le simulazioni tramite la trotterization, si evolve un sistema fisico di $N$ spin che segue il modello di Heisenberg:

$$H_{heis,N} = \sum_{i=0}^{N-1} J(\sigma_i^z\sigma_{i+1}^z + \sigma_i^x\sigma_{i+1}^x + \sigma_i^y\sigma_{i+1}^y)$$

Si noti subito che, nel caso di un sistema a due spin, l'operatore di evoluzione temporale è

$$U(t) = \exp(-iH_{heis,2}^{0,1}t) = \exp(i(\sigma_0^z\sigma_1^z + \sigma_0^x\sigma_1^x + \sigma_0^y\sigma_1^y)t)$$

Tutti gli operatori ad esponente commutano, perciò la decomposizione di Trotter è esatta al primo step, il circuito da applicare allo stato iniziale $|\psi\rangle$, è semplicemente

$$R_{zz}^{0,1}(it)R_{xx}^{0,1}(it)R_{yy}^{0,1}(it)\,|\psi\rangle$$

dove le $R_{\alpha\beta}$ sono implementate come descritto nella Sezione 1.1.3.

Nel caso di 3 spin la situazione è più complessa, infatti l'operatore di evoluzione temporale ha generatori che non commutano:

$$U(t) = \exp(-i(H_{heis,2}^{0,1} + H_{heis,2}^{1,2})t)$$

In questo caso, bisogna applicare l'approssimazione in Equazione 2.2, perciò il singolo Trotter step, è implementato dal circuito:

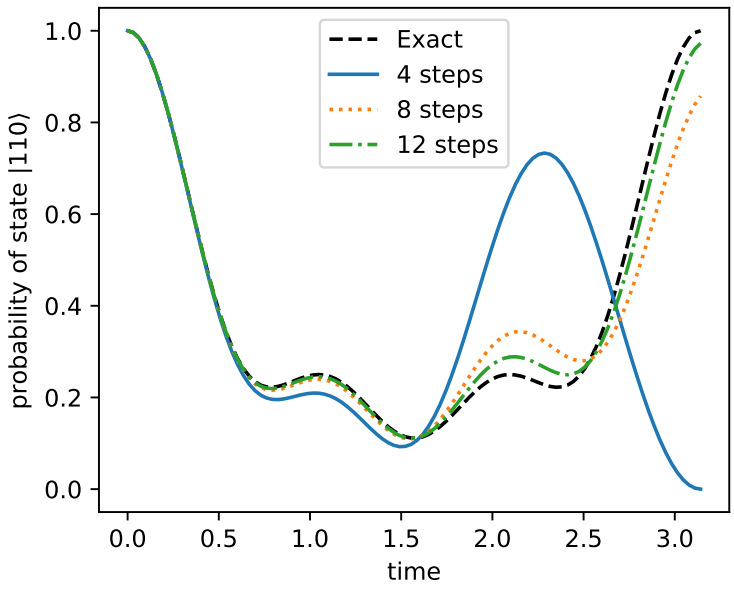
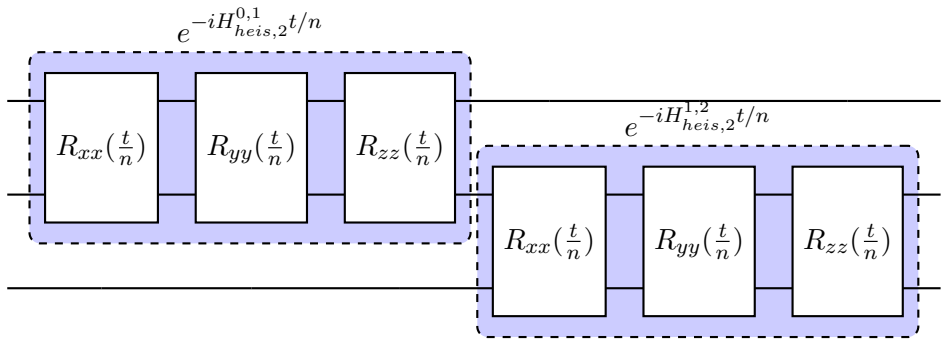



Figura 2.2: Evoluzione di un sistema di tre spin sotto il modello di Ising al variare del numero di trotter steps.




Nella Figura 2.2, viene mostrato come cambia la probabilità di misurare $|110\rangle$ partendo dallo stato $|000\rangle$ al variare del numero di Trotter steps, i.e. variando il numero di volte che applico il circuito precedente a $|000\rangle$. Si osservi che con l'aumento di $n$ la simulazione migliora sensibilmente, ma ogni passo richiede 6 nuovi gate.

## 2.2 Variational quantum simulation

A seconda della fedeltà richiesta nella simulazione, sono necessari un diverso numero di Trotter steps. Ogni passo aumenta la profondità del circuito, il che può velocemente diventare un problema per i device NISQ a causa

dell'incremento del rumore. L'approccio più utilizzato per affrontare questa complicanza è di servirsi dei VQA per risolvere la dinamica. Tale approccio nasce inizialmente in un contesto classico, dove la *simulazione variazionale* è uno strumento necessario ad evitare la crescita esponenziale delle risorse considerando soltanto degli insiemi di prova [42]. Nonostante questo, molti sistemi quantistici riguardanti un grande numero di particelle interagenti non possono comunque essere efficientemente rappresentate con questi metodi, che quindi vengono estesi alla programmazione quantistica.

### 2.2.1 Principi variazionali classici

I più famosi principi variazionali, nati in un contesto classico, usati per la simulazione della dinamica di un sistema sono tre [42]:

- **Principio di Dirac e Frenkel.**
- **Principio di McLachlan.**
- **Time-dependent variational principle.**

Ognuno di questi metodi conduce a un'equazione di evoluzione dei parametri, da risolvere per ottenere la soluzione del problema dinamico.

Si supponga di voler evolvere un certo stato $|\psi(t)\rangle$ e si prenda l'ansatz

$$|\varphi(\theta(t))\rangle$$

dove $\theta$ è un vettore di $p$ parametri che dipendono dal tempo, tale per cui $|\psi(t)\rangle = |\varphi(\theta(t))\rangle$. L'evoluzione temporale di questo stato di un tempo $\delta t$ segue l'equazione di Schrödinger, perciò usando la definizione di differenziale e sostituendo l'ansatz, si ha

$$\frac{\partial}{\partial t}|\psi(t)\rangle = -iH|\psi(t)\rangle \Rightarrow |\psi(t+\delta t)\rangle = |\varphi(\theta(t))\rangle - i\delta t H|\varphi(\theta(t))\rangle$$

Ora, espandendo l'ansatz in serie di Taylor attorno a $t+\delta t$, si ottiene

$$|\varphi(\theta(t+\delta t))\rangle \approx |\varphi(\theta(t))\rangle + \sum_j \frac{\partial|\varphi(\theta(t))\rangle}{\partial\theta_j}\delta\theta_j$$

Siccome l'obbiettivo dei principi variazionali è approssimare l'evoluzione dello stato con l'ansatz a parametri evoluti, si impone che

$$|\psi(t+\delta t)\rangle = |\varphi(\theta(t+\delta t))\rangle$$

Confrontando le due equazioni il problema si riduce a trovare i parametri per cui

$$\sum_j \frac{\partial|\varphi(\theta(t))\rangle}{\partial\theta_j}\delta\theta_j \approx -i\delta t H|\varphi(\theta(t))\rangle \tag{2.3}$$

Per semplificare la notazione, di seguito si userà $\sum_j \frac{\partial|\varphi(\theta(t))\rangle}{\partial\theta_j}\delta\theta_j = |\delta\varphi(\theta(t))\rangle$.

A differenziare i tre algoritmi è l'approccio utilizzato per trovare i parametri che soddisfino la precedente uguaglianza.

### Principio di Dirac e Frenkel

Nell'Equazione 2.3, il termine di sinistra appartiene al sottospazio $\{\frac{\partial|\varphi(\theta(t))\rangle}{\partial\theta_j}\}$ e il termine di destra dovrebbe appartenere al sottospazio ortogonale. Per trovare i parametri corretti si proietta il termine di destra sul sottospazio tangente e si impone che l'overlap sia nullo:

$$\langle\delta\varphi(\theta(t))|\left(\frac{d}{dt}+iH\right)|\varphi(\theta(t))\rangle=0$$

risolvendo si ottiene l'equazione di evoluzione dei parametri:

$$\sum_{k,j}A_{kj}\dot{\theta_k}=-iC_k \tag{2.4}$$

dove

$$A_{kj}=\frac{\partial\langle\varphi(\theta(t))|}{\partial\theta_k}\frac{\partial|\varphi(\theta(t))\rangle}{\partial\theta_j}\qquad C_k=\frac{\partial\langle\varphi(\theta(t))|}{\partial\theta_j}H|\varphi(\theta(t))\rangle$$

### Principio di McLachan

In questo caso si esprime l'Equazione 2.3 come una distanza tra i due termini, l'obiettivo è quella di minimizzarla imponendo

$$\delta||\left(\frac{d}{dt}+iH\right)|\varphi(\theta(t))\rangle||=0$$

Nella computazione classica i parametri sono complessi, in tal caso, l'equazione di evoluzione dei parametri ottenuta risolvendo la precedente è la stessa che si ottiene usando il principio di Dirac e Frenkel.

### Time-dependent variational principle

Questo principio, spesso indicato con l'acronimo TDVP, segue un approccio diverso dai due precedenti, in particolare utilizza il fatto che l'equazione di Schrödingher si ricava a partire della lagrangiana

$$L=\langle\varphi(\theta(t))|\left(\frac{d}{dt}+iH\right)|\varphi(\theta(t))\rangle$$

Applicando le equazioni di Eulero-Lagrange si ottiene l'equazione di evoluzione dei parametri, che nel caso di parametri complessi è uguale alle due precedenti.

### 2.2.2 Principi variazionali quantistici

Giunti a questo punto ci si può chiedere dove, in questi algoritmi classici, entri la computazione quantistica. L'operazione più complessa tra quelle indicate sopra è trovare $A$ e $C$. Anziché calcolarli classicamente, si utilizza un quantum computer per preparare il circuito parametrizzato $|\varphi(\theta(t))\rangle$, che non è altro che l'ansatz già incontrato nella Sezione 1.3, e valutare le entrate di tali oggetti. All'atto pratico, si noti che è possibile scrivere l'ansatz come una sequenza di rotazioni applicate a un certo stato iniziale, generalmente $|0\rangle$, ottendendo

$$|\varphi(\theta)\rangle = R_p(\theta_p) \dots R_1(\theta_1) \, |0\rangle^{\otimes n}$$

e similmente anche la sua derivata.

Nel caso quantistico, i parametri $\theta$ non sono complessi, perciò la forma dell'equazione di evoluzione non è equivalente in tutti e tre i casi, a seconda del problema in esame bisogna scegliere un opportuno principio. Il tema è approfondito in [42].

Nella prossima sezione verrà introdotto un VQA per l'evoluzione dinamica di sistemi quantistici equivalente al principio di McLachlan, che nasce per essere utilizzato su computer quantistici.

## 2.3 Projected Variational Quantum Dynamics

Il *projected-Variational Quantum Dynamics* (pVQD), introdotto per la prima volta in [18], è un metodo variazionale per l'evoluzione dinamica di sistemi quantistici. L'idea alla base dell'algoritmo è quella di dividere l'evoluzione in step temporali infinitesimi e, per ognuno questi, ottimizzare i parametri minimizzando la distanza tra l'evoluzione dell'ansatz tramite un singolo trotter step e l'ansatz con i parametri ottimizzati. Si tratta di un metodo vantaggioso in quanto non utilizza qubit ausiliari, è efficiente ed ottimizza tutti i parametri in un unico passaggio. Il funzionamento schematico dell'algoritmo è quello che segue:

1. **Trovare l'hamiltoniana del problema e tradurla il Pauli gates.** Come avviene per tutti gli algoritmi variazionali, la simulazione può avvenire solo se si riesce a trovare la mappa che permetta di scrivere il problema come circuito. In particolare, bisogna riuscire a scrivere in gate elementari l'operatore di evoluzione temporale, scritto con un singolo trotter step.

2. **Definire un ansatz.** Si vuole cercare un circuito parametrizzato che approssimi lo stato esatto al tempo iniziale. Si tratta di un passaggio fondamentale da cui dipende la fedeltà della simulazione.

3. **Evolvere l'asatz.** Si evolve l'ansatz per un tempo infinitesimo tramite l'operatore di evoluzione precedentemente definito.

4. **Valutare l'overlap** L'idea fondamentale dell'algoritmo è quella di valutare la somiglianza, i.e. l'overlap, tra lo stato ottenuto al punto precedente e l'ansatz con nuovi parametri. Questo per definire la funzione di costo. Si tratta dell'unico passaggio per cui è necessario un quantum computer.

5. **Ottimizzazione dei parametri.** Trovare i parametri che minimizzino la cost function valutata al punto precedente. Una volta noto il gradiente, si può utilizzare un ottimizzatore classico.

6. **Passare allo step temporale successivo.** Si riparte dal punto 3 fino a che si raggiunge il tempo voluto.

L'algoritmo è riassunto nello pseudocodice seguente ed approfondito nelle sezioni successive.

**Algoritmo 4** Pseudocodice del pVQD.

**Require:** $H$ ▷ Hamiltoniana in Pauli stings.
**Require:** $|\varphi(\theta)\rangle$ ▷ Ansatz.
**Require:** $T$ ▷ Durata della simulazione.
**Require:** $\delta t$ ▷ Time step.
**Require:** $\theta_0, d\theta_0 \in \mathbb{R}^p$ ▷ Parametri iniziali.
$t \leftarrow 0$ ▷ Inizializzazione step temporale.
**Require:** $L_{min}$ ▷ Soglia di minimizzazione.
**while** $t < T$ **do**
  $|\varphi(\theta(t+\delta t))\rangle \leftarrow e^{-iH\delta t}\,|\varphi(\theta)\rangle$ ▷ Evoluzione tramite trotter
  $n \leftarrow 0$ ▷ Inizializzazione step ottimizzazione.
  **while** $L(d\theta) < L_{min}$ **do**
    $n \leftarrow n + 1$
    $L(d\theta_n, \delta t) \leftarrow 1 - |\,\langle\varphi(\theta + d\theta)|\varphi(\theta(t+\delta t))\rangle\,|^2$ ▷ Cost function.
    $g_n \leftarrow \nabla_\theta L(d\theta_{n-1}, \delta t)$ ▷ Calcolo gradiente.
    $d\theta_n \leftarrow d\theta_{n-1} - \eta g_n$ ▷ Ottimizzazione parametri (GD).
  **end while**
  $t \leftarrow t + \delta t$
  $\theta(t) \leftarrow \theta + d\theta$ ▷ Aggiornamento parametri.
**end while**
**return** $\theta(T)$

### 2.3.1 Ansatz e cost function

Si consideri un certo stato $|\psi(t)\rangle$ e si prenda l'ansatz $|\varphi(\theta(t))\rangle$, con $\theta$ vettore di dimensione $p$, per cui $|\psi(t)\rangle = |\varphi(\theta(t))\rangle$. Si vuole evolvere lo stato parametrizzato per un tempo infinitesimo $\delta t$. Essendo il tempo piccolo, è sufficiente considerare l'approssimazione dell'operatore di evoluzione al primo trotter

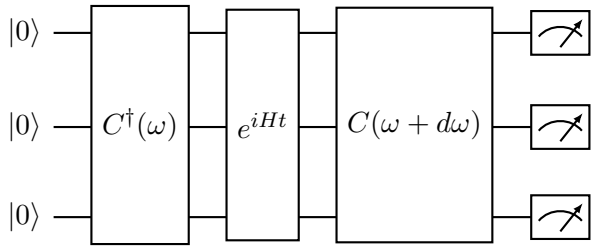



Figura 2.3: Circuito per valutare l'Equazione 2.5.

step:

$$|\varphi(\theta(t+\delta t))\rangle = e^{-iHt}\,|\varphi(\theta(t))\rangle$$

Si definisce la cost funtion

$$L(\theta(t)) = 1 - |\,\langle\varphi(\theta(t+\delta t))|\varphi(\theta+d\theta)\rangle\,|^2 \tag{2.5}$$

si tratta una misura dell'overlap tra lo stato evoluto tramite trotter, a sinistra nel braket, e lo stato con una variazione dei parametri a destra; se i due stati sono ortogonali, ovvero il meno somiglianti possibile, allora la cost funtion vale 1, viceversa se sono uguali, $L$ vale 0. Significa che, per i parametri che meglio approssimano lo stato evoluto, $L$ è minima.

Si può scrivere l'ansatz come un certo insieme di rotazioni $C(\theta)$, applicate ad uno stato iniziale $|0\rangle$, come fatto nella Sezione 2.2.2. Con questa notazione la cost function diventa:

$$L(\theta(t)) = 1 - |\,\langle 0|C^\dagger(\theta)e^{iHt}C(\theta+d\theta)|0\rangle\,|^2 \tag{2.6}$$

### Implementazione della cost function

Una volta definito $C(\theta)$, bisogna misurare la cost function sul quantum computer. Si noti che il modulo quadro nell'Equazione 2.6 non è altro che il valore di aspettazione dell'operatore, noto come *proiettore globale*,

$$P_G = |0\rangle\,\langle 0|$$

sullo stato $C^\dagger(\theta)e^{iHt}C(\theta+d\theta)\,|0\rangle$, cioè la media sul circuito in Figura 2.3. L'operatore $P_g$ viene tradotto in circuito utilizzando i gate di Pauli. Siccome, per il singolo qubit, vale

$$|0\rangle\,\langle 0| = \begin{pmatrix} 1 & 0 \end{pmatrix}\begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} = \frac{1}{2}(I+Z)$$

per un numero arbitrario di qubit, il gate totale sarà il prodotto tensoriale

$$P_G = \frac{1}{2^N}\bigotimes^N (I+Z)$$

L'operatore agisce su tutto lo spazio, ovvero su uno spazio di Hilbert che è esponenzialmente crescente [43]. Il problema con questo tipo di operatori è che potrebbero incontrare un *barren plateau*, ovvero una regione dello spazio in cui il gradiente tende esponenzialmente, nella taglia del sistema, a zero, facendo fallire l'ottimizzazione a causa di un'evoluzione troppo lenta dei parametri. Il pVQD non ha questo problema in quanto valuta la fedeltà tra due stati che distano per una quantità infinitesima [18]. Esiste però una soluzione più generale a questo problema, ed è l'utilizzo dell'operatore, detto *proiettore locale*,

$$P_L = \frac{1}{N}\sum_{j=1}^{N} |0_j\rangle\langle 0_j| \otimes I_{\bar{j}} \tag{2.7}$$

dove $I_{\bar{j}}$ è la matrice identità agente su tutti i qubit eccetto il $j-$esimo, mentre $|0_j\rangle\langle 0_j|$ è il proiettore del qubit $j$. Questo tipo di operatore non cresce esponenzialmente con lo spazio di Hilbert ed ha un gradiente che decresce polinomialmente.

Con questi due proiettori si ottengono le seguenti cost function:

$$L_G = 1 - |0\rangle\langle 0| \qquad L_L = 1 - \frac{1}{N}\sum_{j=1}^{N} |0_j\rangle\langle 0_j| \otimes I_{\bar{j}}$$

L'equivalenza tra le due funzioni deriva dal fatto che, come dimostrato in [43],

$$\langle L_G\rangle = 0 \Leftrightarrow \langle L_L\rangle = 0$$

perciò, nonostante il processo di ottimizzazione possa essere diverso, il minimo è lo stesso. Per studiare meglio le differenze tra le cost function si può calcolare la varianza. Nel caso globale questa è nulla, in quanto è semplicemente la varianza di un proiettore, nel caso locale la situazione è più complessa. Detta $p_j(0)$ la probabilità che il $j-$esimo qubit restituisca 0, la media sullo stato $|\psi\rangle$ è

$$\begin{aligned}\langle L_L\rangle &= \langle\psi| \left(I - \frac{1}{N}\sum_{j=1}^{N} |0_j\rangle\langle 0_j| \otimes I_{\bar{j}}\right) |\psi\rangle \\ &= 1 - \frac{1}{N}\sum_{j=1}^{N} p_j(0)\end{aligned}$$

mentre

$$\begin{aligned}\langle L_L^2 \rangle &= \langle \psi | \left( I - \frac{1}{N} \sum_{j=1}^{N} |0_j\rangle \langle 0_j| \otimes I_{\bar{j}} \right)^2 |\psi\rangle \\ &= 1 + \frac{1}{N^2} \sum_{k,j=1}^{N} c_j c_k^* \langle 0_k | 0_j \rangle - \frac{2}{N} \sum_{j=1}^{N} p_j(0) \\ &= 1 + \frac{1}{N^2} \sum_{j=1}^{N} p_j(0) - \frac{2}{N} \sum_{j=1}^{N} p_j(0)\end{aligned}$$

dove i $c_j$ sono le ampiezze di $|0_j\rangle$ ed è stata usata la condizione di ortonormalizzazione $\langle 0_k | 0_j \rangle = \delta_{kj}$. Dalla definizione di varianza si ha

$$\Delta L_L^2 = \langle L_L^2 \rangle - \langle L_L \rangle^2 = \frac{1}{N^2} \left( \sum_{j=1}^{N} p_j(0) - \left( \sum_{j=1}^{N} p_j(0) \right)^2 \right)$$

Questo risultato implica che in generale la varianza non è nulla, a differenza di quanto avviene per la funzione globale.

Una possibile implementazione in Qiskit della misura della cost function è quella che segue, dove viene utilizzata la primitiva `Estimator()` per calcolare il valore di aspettazione del proiettore. I risultati del confronto delle performance dei due operatori si trovano alla Sezione 3.2.

```
circuit = QuantumCircuit(N)
circuit = circuit.compose(ansatz(θ))
circuit = circuit.compose(trotter_step.inverse())
circuit = circuit.compose(ansatz(θ + dθ))

results = estimator.run(circuit, projector).result()
L = 1 - results
```

### 2.3.2 Ottimizzazione

L'evoluzione dei parametri avviene cercando la variazione $d\theta$ che minimizza $L$. Il vantaggio nell'ottimizzare questa quantità anziché i parametri sta nel fatto che, come verrà dimostrato empiricamente nella Sezione 3.2, è possibile usare il risultato dell'ottimizzazione precedente per velocizzare il processo.

Nella pratica, l'ottimizzazione avviene ricavando il gradiente tramite la parameter-shift rule (Sezione 1.3.1), dove la cost funtion viene calcolata come precedentemente detto. Una volta noto il gradiente ci si affida ai computer classici sfruttandone la potenza per calcolare i nuovi parametri.

Quest'ultimo passaggio può avvenire, ad esempio, con uno degli algoritmi di ottimizzazione citati nella Sezione 1.3.1: GD, ADAM o SPSA. Il confronto tra questi algoritmi può essere trovato nella Sezione 3.2.

# Capitolo 3

# Evoluzione dinamica del modello di Ising

Nel seguente capitolo si introduce (Sezione 3.1) un modello di evoluzione con cui testare il pVQD. Dunque (Sezione 3.2) vengono raccolti i risultati ottenuti dalle simulazioni, prima su backend reali, quindi su backend rumorose. L'algoritmo è stato implementato tramite Qiskit. Il codice sorgente utilizzato per le simulazioni si trova su [44].

## 3.1 Modello di evoluzione

I VQA, sono pensati per funzionare con una vasta gamma di problemi, ma in questa tesi ci si concentrerà su uno specifico modello per studiare il comportamento del pVQD. Si immagini di avere una catena aperta di $N$ fermioni in una dimensione, tali che abbiano spin parallelo all'asse $z$ e che l'interazione avvenga solo tra coppie di particelle vicine. Inoltre, si accende un campo magnetico lungo $x$. Questo sistema risponde al *Transverse Field Ising Model*:

$$H = J \sum_{i=0}^{N-1} \sigma_i^z \sigma_{i+1}^z + h \sum_{i=0}^{N} \sigma_i^x$$

dove $J$ è una misura dell'interazione tra gli spin, mentre $h$ l'intensità del campo esterno. Nelle successive simulazioni si userà $J = 1/4$ e $h = 1$. La catena avrà $N = 3$ spin.

### 3.1.1 Ansatz

Per la simulazione del modello di Ising è stato scelto un ansatz nella forma

$$C(\theta) = \prod_{l=1}^{d} \left[ \prod_{i=1}^{N} R_\alpha^i(\theta_{i,l}) \prod_{j=1}^{N-1} R_{zz}^{j,j+1}(\theta_{j,l}) \right] \tag{3.1}$$

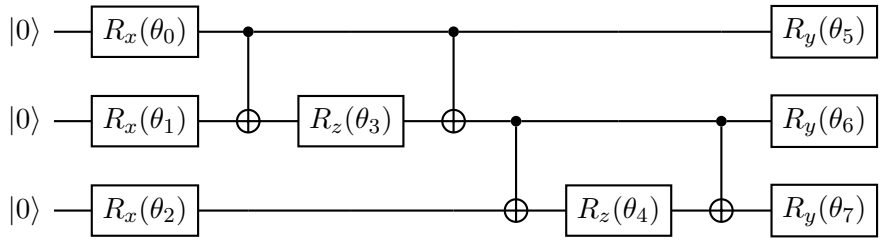



Figura 3.1: Circuito che implementa l'ansatz con una profondità 1 per 3 spin. Aumentare la profondità significa ripetere $d$ volte questo set di gate. Si noti anche l'implementazione di $R_{zz}(\theta)$ tramite i due CNOT (Sezione 1.1.3).

dove $d$ è detta *profondità del circuito*, un suo aumento offre una maggiore precisione nell'approssimazione dello stato a costo di un incremento della lunghezza del circuito. Si sceglie $R_\alpha = R_x$ se $l$ è pari, mentre $R_\alpha = R_y$ se $l$ è dispari. Le rotazioni sono stati implementate come spiegato nella Sezione 1.1.3, il circuito che ne risulta per una profondità di $d = 1$ è in Figura 3.1. Con questo ansatz, il numero di parametri da ottimizzare è

$$p = N(2d + 1) - d$$

### 3.1.2 Trotter step

Una volta definito l'algoritmo bisogna implementare l'evoluzione tramite Trotter. L'operatore di evoluzione temporale nel caso del modello in questione con tre spin è

$$U(t) = \exp\left(-ih(\sigma_0^x + \sigma_1^x + \sigma_2^x)\delta t - iJ(\sigma_0^z\sigma_1^z + \sigma_1^z\sigma_2^z)\delta t\right)$$

Le rotazioni che coinvolgono l'asse $x$ sono applicate a spin diversi, perciò commutano tra loro, mentre quelle che coinvolgono $z$ no. Significa che la decomposizione di Trotter è approssimata, siccome il tempo è infinitesimo, è sufficiente fermarsi a $n = 1$. Il circuito che implementa l'evoluzione, i.e. $e^{-itH}$, è in Figura 3.2.

### 3.1.3 Misura delle osservabili

Una volta ottenuti i parametri che approssimano lo stato al tempo $t + \delta t$ si possono valutare le osservabili di interesse in modo da confrontarle con i risultati attesi. In particolare, si vuole misurare la magnetizzazione totale, sia sull'asse $x$ che sull'asse $z$:

$$\langle\sigma_\alpha\rangle = \frac{\sum_i^N \sigma_i^\alpha}{N}$$

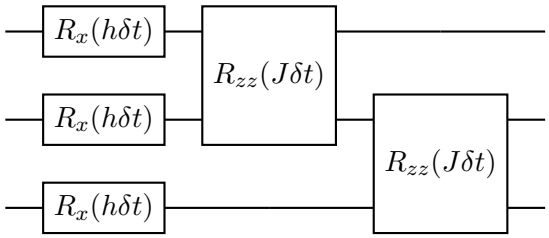



Figura 3.2: Circuito che implementa il singolo Trotter step.

dove $N$ è il numero di spin e $\alpha \in \{x, z\}$.

La misura avviene calcolando il valore di aspettazione di tali osservabili sul circuito $C(\theta)\,|0\rangle$ (Equazione 3.1), dove $\theta$ sono i parametri ottimizzati per il dato step temporale. Per ottenere tale valore tramite Qiskit, si usa la primitiva `Estimator()`.

## 3.2 Risultati

Con lo scopo di studiare il comportamento del pVQD, si compiono le misure utilizzando i simulatori di quantum computer resi disponibili da Qiskit. Questo permette di eseguire i circuiti in modo ideale, oppure con dei modelli di rumore artificiali. In una prima parte si useranno simulatori, in modo da valutare quale sia il comportamento dell'algoritmo, per poi passare a simulazioni rumorose, in modo da studiare la robustezza agli errori e le strategie per migliorarla.

### 3.2.1 Simulatori ideali

#### Studio dello shot noise

Limitandosi all'utilizzo di backend prive di rumore, l'unico errore presente è lo *shot noise*, errore dovuto alla natura probabilistica della computazione quantistica. La probabilità di misurare un certo stato è infatti valutata come

$$P = \frac{n_{successi}}{n_{shots}}$$

Non disponendo, ovviamente, di un numero infinito di misure del circuito ($n_{shot}$), la precisione con cui si valuta la probabilità è limitata. Attraverso Qiskit è possibile trovare i valori di aspettazione a meno dello shot noise con la simulazione tramite vettori di stato (Statevector simulation, Sezione 1.1.6), cioè computando direttamente i vettori e le matrici che rappresentano stati e operatori[1]. Confrontando i risultati della simulazione tramite

[1]Si tratta di un metodo non efficiente.

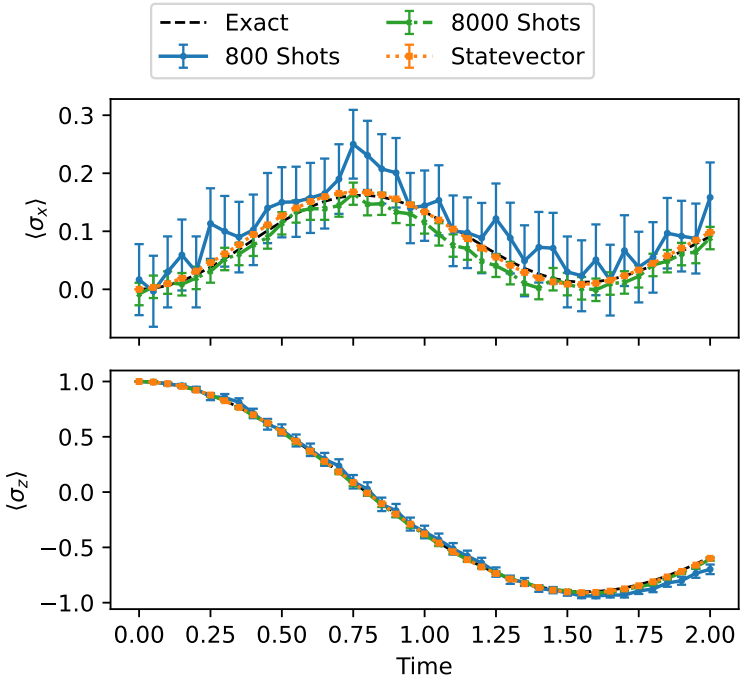



Figura 3.3: Confronto tra la simulazione tramite statevector (senza shot), la simulazione con shots ed il valore atteso.

questa backend e quelli ottenuti con Aer (Sezione 1.1.6), si riesce a valutare l'impatto dell'errore statistico. In Figura 3.3 sono raccolti i risultati della simulazione del modello di Ising utilizzando la cost function globale e l'ottimizzatore GD. La prima informazione che si può trarre dal grafico è che, come è ragionevole, la simulazione senza shot segue meglio la distribuzione, gli unici errori in tale caso sono, infatti, le approssimazioni dovute all'ansatz e al Trotter step. Nel caso della simulazione con shot, si nota che anche con poche esecuzioni del circuito la magnetizzazione misurata lungo l'asse $z$ segue molto bene la simulazione esatta; mentre la magnetizzazione lungo l'asse $x$ è più difficile da valutare. Quest'ultimo punto è conseguenza dell'ansatz scelto, che riesce a catturare meglio la prima osservabile, in ogni caso aumentando gli shot la simulazione riesce a simulare bene anche l'andamento di $\sigma_x$. In secondo luogo si può anche osservare che con l'aumento del tempo le curve iniziano a divergere da quella esatta, circa da $t = 1.75$. Tra le possibili cause c'è il fatto che con il procedere della simulazione, l'entanglement tra gli spin del sistema aumenta e l'ipotesi per cui l'interazione è solo locale diventa meno corretta. Inoltre, con ogni step temporale l'approssimazione dovuta al Trotter step si accumula.

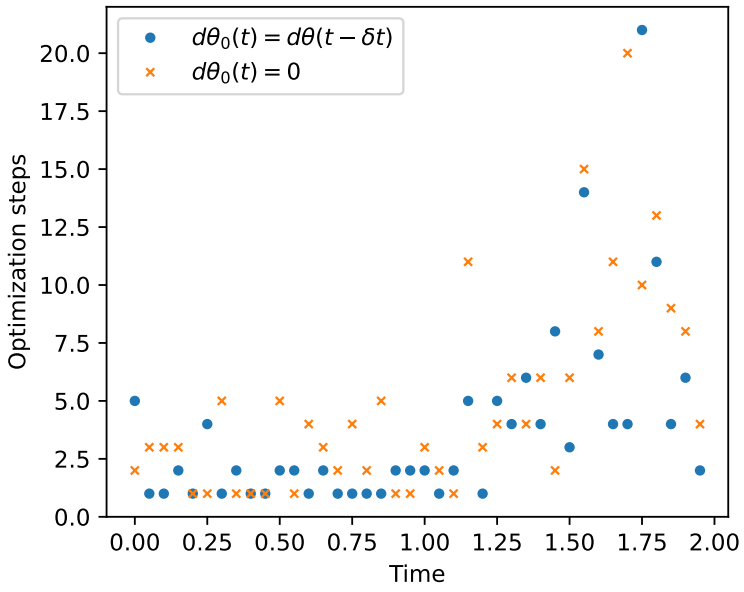



Figura 3.4: Confronto tra step di ottimizzazione richiesti nel caso in cui si parta dallo shift precedente oppure si parta sempre da uno shift nullo. L'ottimizzazione è stata eseguita tramite GD.

### Studio dell'ottimizzazione

Per studiare le performance dell'algoritmo è bene trovare quanti step di ottimizzazione siano necessari per minimizzare la funzione di costo.

Come prima cosa si valuta quale sia lo shift dei parametri con cui è meglio partire, in Figura 3.4 vengono mostrati gli step di ottimizzazione richiesti ad ogni passo temporale per minimizzare $L$. Risulta che partendo dallo shift ottimizzato nello step precedente, il numero medio di step richiesti sia di 3.7, partendo invece da uno shift iniziale pari a 0, la media degli step sia 4.9. Queste quantità servono solo per una stima qualitativa, in quanto il valore è fortemente condizionato dalla lunghezza del tempo di simulazione. Nelle successive simulazioni verrà utilizzato il risultato dell'ottimizzazione precedente.

Sempre nella Figura 3.4 si può notare che, coerentemente con quanto detto per la Figura 3.3, gli stati a tempi maggiori sono più difficili da catturare sempre a causa dell'aumento dell'entanglement tra gli spin.

Una possibile strategia per affrontare questo problema è l'utilizzo di ADAM in quanto adatta il peso del gradiente usando le informazioni degli step di ottimizzazione precedenti, nei passaggi più difficili dovrebbe riuscire a riscalare il gradiente in maniera più precisa. Perciò si procede confrontando l'ottimizzazione degli ottimizzatori GD, ADAM e SPSA, si veda Sezione 2.3.2 per la teoria. Dai risultati, raccolti nella Figura 3.5, si evince che per

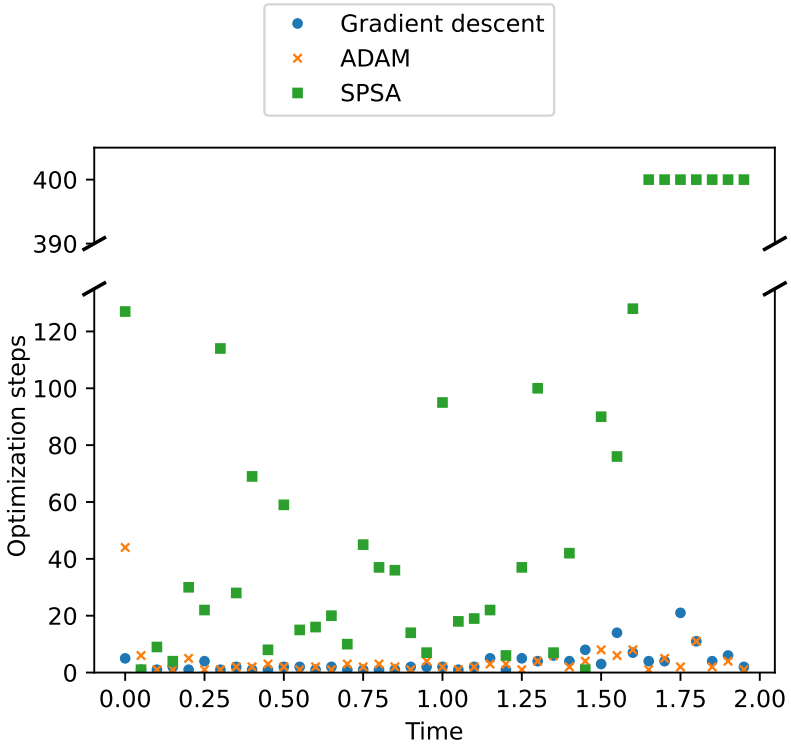



Figura 3.5: Step di ottimizzazione per GD, ADAM e SPSA.

GD e ADAM gli step di ottimizzazione necessari sono in media 4. Si noti che ad alti tempi ADAM inizia ad essere più conveniente rispetto al GD come precedentemente ipotizzato. Per quanto riguarda SPSA, l'ottimizzazione inizia a fallire per tempi grandi e gli step di ottimizzazione necessari sono sensibilmente più alti (35, limitatamente alla regione in cui l'ottimizzazione ha successo). Questi risultati sono facilmente spiegabili analizzando il funzionamento dell'algoritmo, siccome SPSA sfrutta una versione approssimata del gradiente, gli errori si accumulano con il procedere della simulazione e così la fedeltà dei risultati peggiora; inoltre, sono necessarie solo due misure del circuito per ogni step, questo a costo di un maggior numero di ottimizzazioni.

Siccome gli step di ottimizzazione richiedono un diverso numero di misure del circuito a seconda dell'algoritmo scelto, per studiare la loro efficienza su un computer quantistico conviene piuttosto valutare il numero di esecuzioni del circuito. Per ogni step di ottimizzazione sono necessarie $2p$ misure per GD e ADAM e 2 per SPSA, dove $p$ è il numero di parametri. Quindi, la media di esecuzioni del circuito per step di ottimizzazione è di 70 per SPSA e 150 per GD e ADAM.

Si conclude che per misure su di hardware reale l'ottimizzatore migliore è

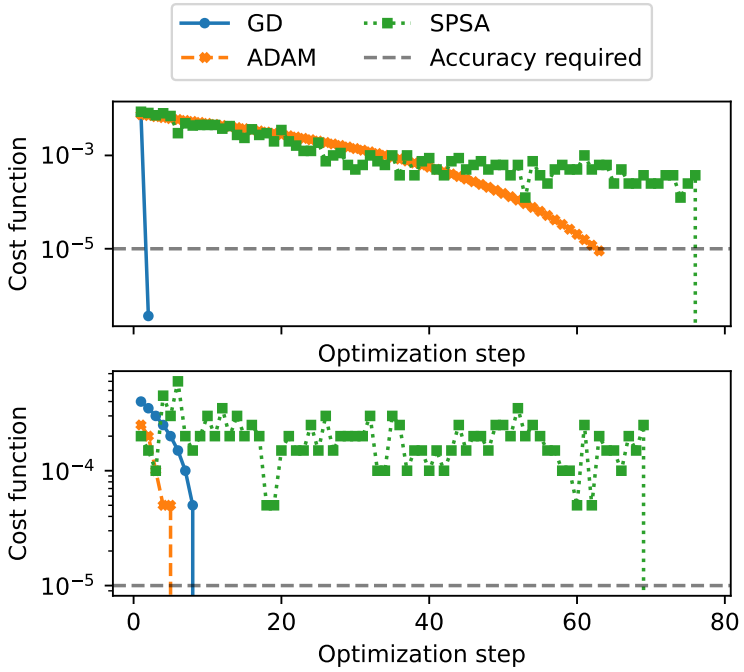



Figura 3.6: Curva di ottimizzazione con i vari ottimizzatori, si noti che l'asse delle y è in scala logaritmica. In alto $t = 0.05$, mentre in basso $t = 1.5$.

SPSA, infatti, limitatamente a pochi time step, gli errori dovuti all'approssimazione del gradiente dovrebbero essere compensati dal guadagno nella diminuzione dei circuiti, e quindi degli errori, da eseguire.

Per uno studio più approfondito si valuta la curva di ottimizzazione, cioè il valore della cost function negli step, Figura 3.6, a diversi tempi. Si può notare il comportamento stocastico di SPSA, infatti a differenza degli altri due ottimizzatori, ad ogni step non va in direzione del minimo, ma il valore di $L$ cala solo definitivamente.

A riprova del fatto che SPSA è il metodo meno accurato, si noti che l'ultimo step cambia la cost function molto di più di quanto lo facciano gli altri. Spesso in questi casi, l'ottimizzazione non è consistente e procedendo con un ulteriore step il valore della funzione di costo potrebbe tornare sopra il minimo. Si tratta di un effetto dovuto all'approssimazione del gradiente.

### Studio della cost function

Nella Sezione 2.3 sono stati introdotti due diversi operatori per calcolare la cost function, il proiettore globale, usato fino ad ora, e il proiettore locale. In questa parte ci si occuperà del confronto tra le due implementazioni di $L$.

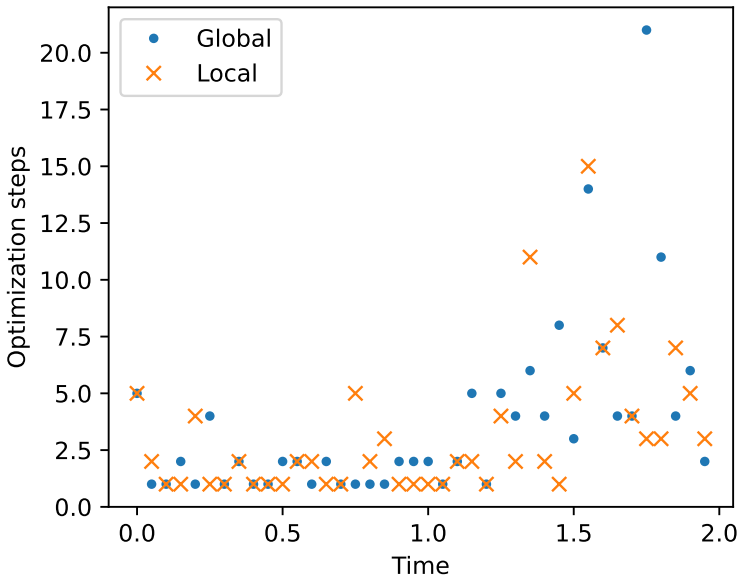



Figura 3.7: Confronto tra gli step di ottimizzazione necessari con la cost function locale e quella globale.

In Figura 3.7, vengono rappresentati gli step di ottimizzazione nel tempo. In generale, si hanno risultati confrontabili in entrambi i casi.

### 3.2.2 Simulatori con rumore

Una volta studiato il pVQD nel caso ideale, si vuole capire quali siano i limiti di funzionamento in presenza di rumore. Si tratta di un'analisi che ha lo scopo di capire quali siano le migliori strategie per rendere l'algoritmo sufficientemente robusto agli errori perché possa essere eseguito sull'hardware reale.

Il primo modello di rumore applicato ricalca il rumore del quantum computer di IBM "imbq_lagos", ci si sta dunque ponendo in condizioni simili al caso reale. In queste condizioni, l'ottimizzazione fallisce, sia con che senza l'utilizzo di ZNE (Sezione 1.2.2). Perciò vengono costruiti semplici modelli di rumore per poter variare i parametri e le tipologie degli errori.

#### Curve di ottimizzazione

In assenza di algoritmi di mitigazione, vengono costruite le curve di ottimizzazione per il primo step temporale. Il modello di rumore utilizzato comprende errori di bit flip e di phase-amplitude damping. Per minimizzare le imprecisioni dovute allo shot noise, si sceglie $n_{shots} = 10^6$, in questo modo si può approssimare il backend a statevector simulator.

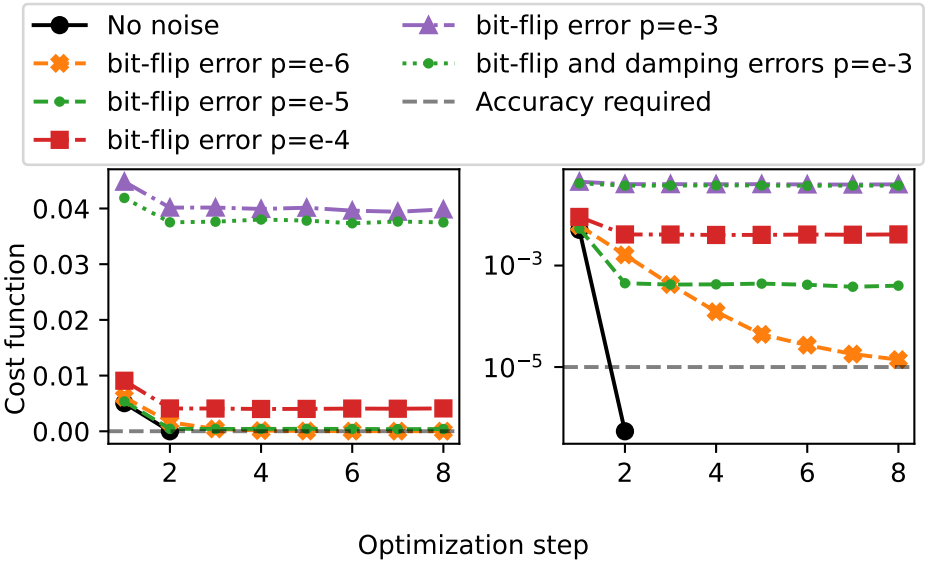



Figura 3.8: Curve di ottimizzazione per $t = 0.05$ a diverse intensità di rumore. L'ottimizzatore utilizzato è il GD. A destra in scala normale, a sinistra in scala logaritmica.

Come si nota dal grafico in Figura 3.8, il rumore introduce un bias nella valutazione della cost function. Infatti, dalla definizione (Equazione 2.5), $L$ è definita positiva, perciò tutti gli errori commessi non potranno che allontanarla dal minimo. Inoltre, la pendenza della curva è indipendente dal livello di rumore, ciò significa che l'ottimizzazione dei parametri funziona correttamente, ma la threshold non può essere raggiunta. Si può inoltre notare che gli errori di damping diminuiscono il bias anziché aumentarlo. Questo è dovuto per lo più al damping dell'ampiezza. Infatti, questo tipo di errore lascia invariato l'autostato $|0\rangle$, in quanto corrisponde allo stato con energia nulla, mentre la probabilità di ottenere 1 cala. Siccome la cost function è il valore di aspettazione di un proiettore su $|0\rangle^{\otimes N}$, è come trovare il numero di volte che la misura del circuito restituisce l'autovalore $0^{\otimes N}$, questo valore aumenta con gli errori di damping, quindi $1 - \langle P_G \rangle$ è in media più basso.

### Calcolo del bias

Per valutare il bias introdotto, si ottimizzano i parametri utilizzando una backend ideale, ma una volta raggiunto il minimo si calcola la cost function con un modello di rumore. In questo modo, infatti, si sta misurando una funzione il cui valore dovrebbe 0, entro l'accuratezza richiesta, perciò qualsiasi valore non nullo ottenuto è una misura del rumore. In Figura 3.9 vengono raccolti i risultati di questa analisi. Il grafico in questione è utile,

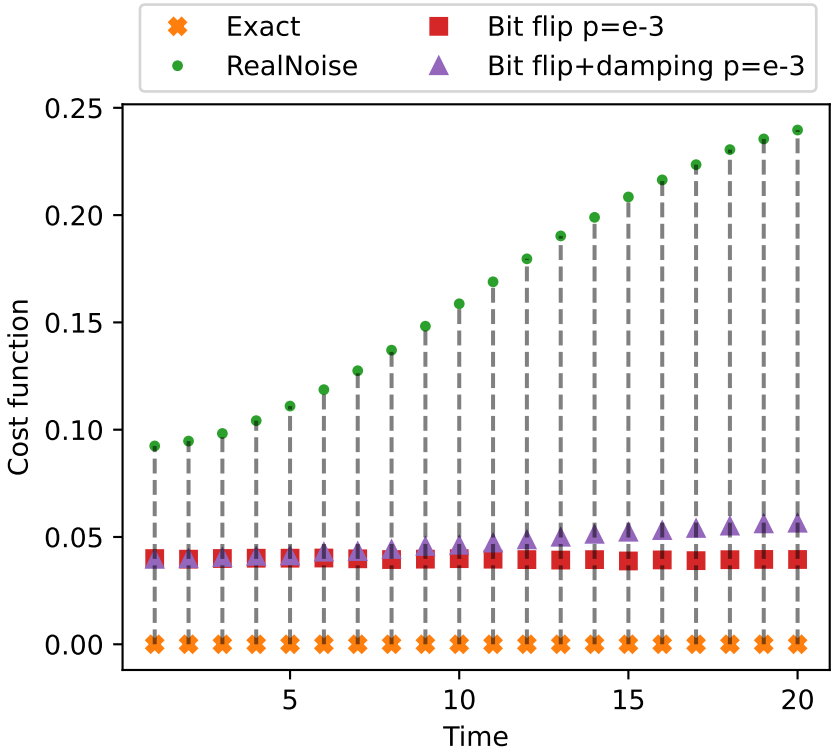



Figura 3.9: Valore minimo della cost function globale ottimizzata in modo esatto, ma misurata in presenza di rumore. A causa della scala del grafico sembra che la $L$ esatta sia nulla, in realtà è solo minore di $10^{-5}$.

oltre che per avere una stima del bias introdotto dal rumore, per osservare quali siano gli effetti delle singole tipologie di errore. Innanzi tutto si osservi che l'errore di bit flip introduce un bias indipendente dal time step. Infatti, questo errore non fa altro che rimappare $|0\rangle \mapsto |1\rangle$ e viceversa, perciò i suoi effetti dipendono solo dalla probabilità con cui questo avviene, indipendentemente da altri fattori, quali lo stato iniziale o il tempo di simulazione. Invece, per quanto riguarda il phase-amplitude damping, il bias non è costante, ma aumenta col tempo. Il motivo di quest'ultimo effetto è che si parte da uno stato con autovalori nulli, per poi applicare una serie di rotazioni che lo allontanano mano a mano dal ground state, con l'introduzione di fasi e con l'aumento della probabilità di $|1\rangle$; gli errori di damping smorzano fase e ampiezza, perciò il loro effetto sulla computazione va via a via aumentando e così l'overlap diminuisce.

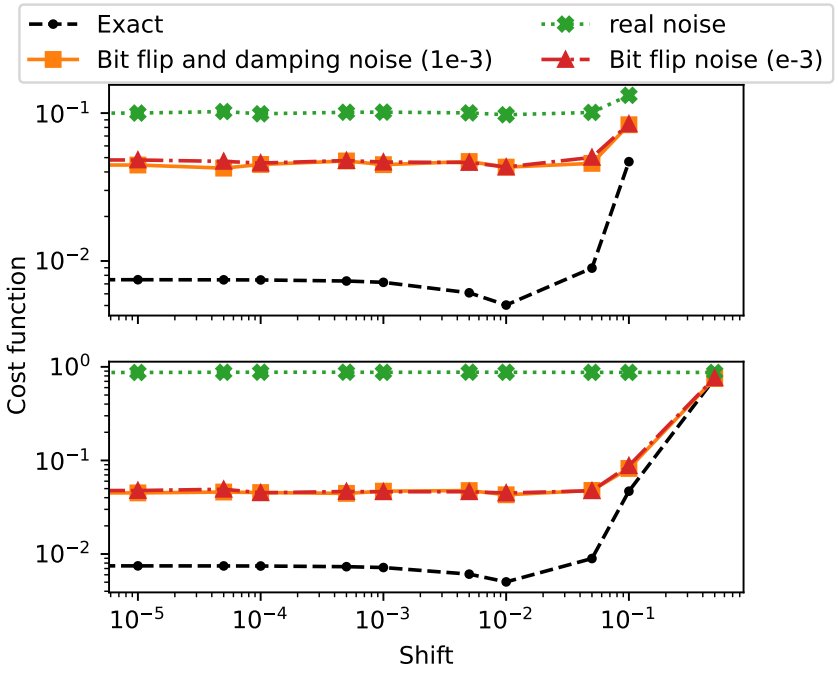



Figura 3.10: Cost function al variare degli shift a diverse intensità di rumore. Sopra a $t = 0.05$, sotto a $t = 0.3$.

### Sensibilità alla variazione dei parametri

Oltre che introdurre un bias assoluto nella misura della cost function, il rumore diminuisce la sensibilità alla variazione dei parametri. Di fatto si tratta dello stesso effetto, ma osservato da due punti di vista differenti, nel primo caso ci si concentra sull'irraggiungibilità del minimo, nel secondo sull'impossibilità di reagire allo shift dei parametri in quanto la variazione della cost function sarebbe molto minore del rumore. In Figura 3.10, vengono mostrati i valori di $L$ al variare degli shift a diverse intensità di rumore. Dal grafico si può notare che per variazioni troppo piccole, la sensibilità dei simulatori non è sufficiente a mostrare alcun effetto sulla funzione di costo. Quando il valore degli shift è nell'ordine di grandezza del gradiente (cioè $10^{-2}$), allora la $L$ inizia a calaare. Quando i parametri aumentano troppo, l'overlap inizia a diminuire molto rapidamente.

Al primo step temporale il rumore ha un effetto solo parziale, nonostante la funzione di costo non cali in modo vistoso, rimane comunque che se si procede nella direzione sbagliata, $L$ aumenta. Seppur con minore precisione l'ottimizzazione è ancora possibile.

Quando si ripetono le misure a un tempo maggiore, nel caso del rumore reale, la cost function non sembra subire alcun effetto nemmeno a seguito

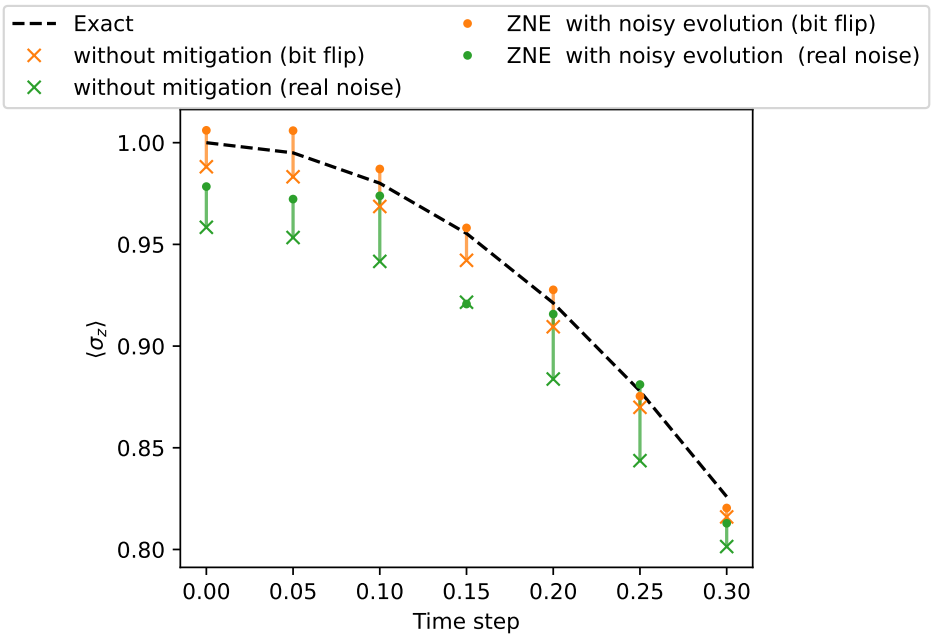



Figura 3.11: Confronto tra le misure con e senza ZNE a diverse intensità di rumore.

di una grande variazione dei parametri. Il bias è più alto di quanto gli stati vengano allontanati. L'ottimizzazione in questo caso è impossibile.

Le osservazioni appena compiute spiegano perché anche se l'ottimizzazione fallisce, per i primi step temporali le osservabili sembrano seguire i valori attesi anche in presenza di noise reale.

### Effetti dello ZNE

Il primo passo per affrontare il rumore è quello di introdurre un algoritmo di mitigazione. Per questa tesi si è scelto di implementare la ZNE (Sezione 1.2.2). Il codice viene scritto in Python utilizzando il già citato Mitiq. Per introdurre il rumore si utilizza la tecnica dell'unitary folding.

Per un'analisi più accurata lo ZNE viene introdotto per gradi. Si inizia ottimizzando in modo esatto la cost function e facendo una misura rumorosa della magnetizzazione. In Figura 3.11, viene mostrato il variare di $\langle \sigma_z \rangle$ nel tempo in presenza di bit flip e rumore reale, con e senza mitigazione del rumore. Il grafico evidenzia un chiaro miglioramento dei valori di aspettazione se si utilizza lo ZNE. Questo a prova del funzionamento di tale algoritmo.

Si procede introducendo il rumore anche nell'ottimizzazione della funzione di costo. Quello reale è ancora troppo alto perché avvenga correttamente l'ottimizzazione, perciò si continuano ad usare modelli costruiti apposita-

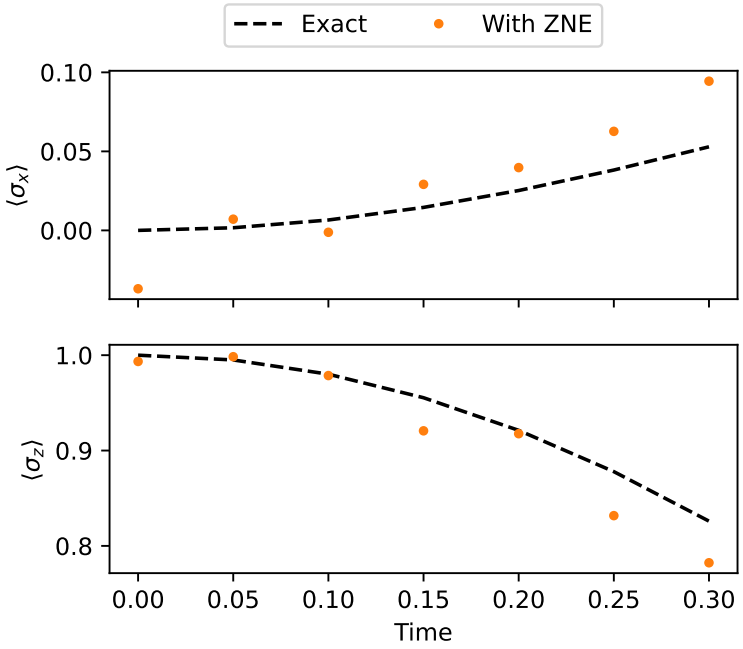



Figura 3.12: Simulazione in presenza di rumore con ZNE.

mente. Viene dunque simulato l'andamento della magnetizzazione in $x$ e $z$ in presenza di un modello di rumore contenente il bit flip e il phase-amplitude damping error. La probabilità di commettere errore di bit flip è nell'ordine di quella reale, perciò di $10^{-3}$. I coefficienti che determinano l'intensità del secondo tipo di errore sono compatibili con quelle descritte in [32].

In queste condizioni l'ottimizzazione ha successo, i valori di aspettazione sono raccolti in Figura 3.12. Dai risultati si nota che $\langle \sigma_z \rangle$ viene catturata molto bene, mentre $\langle \sigma_x \rangle$ è meno fedele a quanto atteso; si tratta però di un comportamento coerente a quanto ottenuto anche in assenza di rumore, ed è dovuto alla forma dell'ansatz.

# Capitolo 4

# Conclusioni

## 4.1 Considerazioni sui risultati

I risultati ottenuti dalle esecuzioni su simulatori ideali (Sezione 3.2.1) mostrano la convergenza dei risultati del pVQD e forniscono una prova del funzionamento dell'algoritmo indipendente da fattori esterni, quali hardware o errori statistici (shot noise). Si veda in particolare la Figura 3.3. Ulteriori informazioni che si possono ricavare da simulazioni ideali riguardano l'efficienza del pVQD. A parte l'efficienza classica, si è interessati all'unico processo che necessita di misure sul quantum computer, cioè la valutazione della cost function. Per questo viene esaminato il processo di ottimizzazione confrontando diversi tipi di ottimizzazione (Figura 3.5) e concludendo che per computer NISQ il miglior algoritmo è SPSA. Da un confronto tra un approccio globale e uno locale (Figura 3.7), non risultano particolari differenze ne nel processo di ottimizzazione ne sul comportamento complessivo dell'algoritmo.

Dunque si introducono dei modelli di rumore nelle simulazioni. Come è evidente dallo schema di funzionamento tipico dei VQA l'unico processo fortemente influenzato dal rumore è l'ottimizzazione, la quale fallisce con modelli realistici. Introducendo mano a mano errori con probabilità crescenti si nota la presenza di un bias (Figura 3.8) che nasconde il minimo della funzione rendendolo inaccessibile. Questo bias viene misurato numericamente (Figura 3.9) e si conclude che diversi tipi di errore vi incidono in modo differente. Un secondo effetto è quello di diminuire la sensibilità alle variazioni dei parametri, infatti, perché l'ottimizzazione possa avvenire, bisogna essere sufficientemente precisi da misurare una differenza nell'overlap nel momento in cui i parametri cambiano. Se il bias è molto maggiore rispetto a quanto gli shift aumentino (o diminuiscano) la cost function, allora questa non può avvenire (Figura 3.10).

Infine, si valuta il comportamento dell'algoritmo di mitigazione del rumore ZNE. L'ottimizzazione fallisce con un modello reale. Costruendo un

modello di rumore solo con bit flip e phase amplitude damping error, a rate paragonabili a quelli reali, la simulazione, grazie allo ZNE, ha successo (Figura 3.12).

## 4.2 Prospettive future

Con la ricerca che si focalizza sul raggiungimento della *quantum advantage* [11] su computer NISQ, la simulazione dei fenomeni fisici è un campo molto studiato. Infatti, sono stati ottenuti risultati che si avvicinano di molto a quelli ottenuti con supercomputer classici. Si veda, ad esempio, [13] dove il modello di Ising è stato simulato con successo per un sistema di 127 spin usando la Trotter evolution (Sezione 2.1) e lo ZNE. I VQA sono sicuramente una delle strade percorribili per il raggiungimento di questo obbiettivo.

In particolare, il pVQD ha mostrato di poter essere utilizzato in presenza di rumore blando con solo una semplice tecnica di mitigazione. Perchè possa essere eseguito con successo su hardware reale bisogna aumentarne la robustezza agli errori. Un primo passo per farlo è sicuramente l'implementazione di algoritmi di mitigazione più performanti, come la PEC, e che includano anche errori di misura [35]. Un secondo passo è la riduzione della lunghezza dei circuiti calcolando la cost function con metodi alternativi, come le Classical Shadows [45, 46] o l'Hadamard test [47]. In ogni caso, le strategie da utilizzare dipenderanno anche dall'architettura dell'hardware.

# Bibliografia


[1] Paul Benioff. ≪The computer as a physical system: A microscopic quantum mechanical Hamiltonian model of computers as represented by Turing machines≫. In: *Journal of Statistical Physics* 22 (mag. 1980), pp. 563–591. DOI: 10.1007/BF01011339.

[2] Richard P. Feynman. ≪Simulating physics with computers≫. In: *International Journal of Theoretical Physics* (giu. 1982). DOI: 10.1007/BF02650179.

[3] David Deutsch e Richard Jozsa. ≪Rapid Solution of Problems by Quantum Computation≫. In: *Proceedings of the Royal Society of London Series A* 439.1907 (ott. 1992), pp. 553–558. DOI: 10.1098/rspa.1992.0167.

[4] P.W. Shor. ≪Algorithms for quantum computation: discrete logarithms and factoring≫. In: *Proceedings 35th Annual Symposium on Foundations of Computer Science*. 1994, pp. 124–134. DOI: 10.1109/SFCS.1994.365700.

[5] David P. DiVincenzo. ≪The Physical Implementation of Quantum Computation≫. In: *Fortschritte der Physik* 48.9-11 (set. 2000), pp. 771–783. DOI: 10.1002/1521-3978(200009)48:9/11<771::aid-prop771>3.0.co;2-e.

[6] Francesco Tacchino et al. ≪Quantum Computers as Universal Quantum Simulators: State-of-the-Art and Perspectives≫. In: *Adv. Quantum Technologies* 3.3 (dic. 2019), p. 1900052. DOI: 10.1002/qute.201900052.

[7] *IBM Quantum*. URL: %5Curl%7Bhttps://quantum-computing.ibm.com/%7D.

[8] Google LLC. *Google Quantum AI*. URL: https://quantumai.google/.

[9] *IonQ*. URL: https://ionq.com/.

[10] *Quantinuum*. URL: https://www.quantinuum.com/.

[11] John Preskill. *Quantum computing and the entanglement frontier - Rapporteur talk at the 25th Solvay Conference*. 2012. arXiv: 1203.5813 [quant-ph].

[12] John Preskill. ≪Quantum Computing in the NISQ era and beyond≫. In: *Quantum* 2 (ago. 2018), p. 79. DOI: 10.22331/q-2018-08-06-79.

[13] Kim Youngseok et al. ≪Evidence for the utility of quantum computing before fault tolerance≫. In: *Nature* 618 (giu. 2023). DOI: 10.1038/s41586-023-06096-3.

[14] M. Cerezo et al. ≪Variational quantum algorithms≫. In: *Nature Reviews Physics* 3.9 (ago. 2021), pp. 625–644. DOI: 10.1038/s42254-021-00348-9.

[15] Kishor Bharti et al. ≪Noisy intermediate-scale quantum algorithms≫. In: *Reviews of Modern Physics* 94.1 (feb. 2022). DOI: 10.1103/revmodphys.94.015004.

[16] Edward Farhi, Jeffrey Goldstone e Sam Gutmann. *A Quantum Approximate Optimization Algorithm*. 2014. arXiv: 1411.4028 [quant-ph].

[17] Alberto Peruzzo et al. ≪A variational eigenvalue solver on a photonic quantum processor≫. In: *Nature Communications* 5.1 (lug. 2014). DOI: 10.1038/ncomms5213.

[18] Stefano Barison, Filippo Vicentini e Giuseppe Carleo. ≪An efficient quantum algorithm for the time evolution of parameterized circuits≫. In: *Quantum* 5 (lug. 2021), p. 512. DOI: 10.22331/q-2021-07-28-512.

[19] Michael A. Nielsen e Isaac L. Chuang. *Quantum Computation and Quantum Information*. 10ª ed. Cambrige university press, 2010.

[20] Thomas G. Wong. *Introduction to Classical e Quantum Computing*. 1ª ed. Rooted groove, 2022.

[21] Elias Fernandez-Combarro Alvarez. *A practical introduction to quantum computing: from qubits to quantum machine learning and beyond*. 2020.

[22] Adriano Barenco et al. ≪Elementary gates for quantum computation≫. In: *Physical Review A* 52.5 (ott. 1995), pp. 3457–3467. DOI: 10.1103/physreva.52.3457.

[23] P. Oscar Boykin et al. *On Universal and Fault-Tolerant Quantum Computing*. 1999. arXiv: quant-ph/9906054 [quant-ph].

[24] Qiskit contributors. *Qiskit: An Open-source Framework for Quantum Computing*. 2023. DOI: 10.5281/zenodo.2573505.

[25] Ying Li e Simon C. Benjamin. ≪Efficient Variational Quantum Simulator Incorporating Active Error Minimization≫. In: *Phys. Rev. X* 7 (2 giu. 2017), p. 021050. DOI: 10.1103/PhysRevX.7.021050.

[26] Kristan Temme, Sergey Bravyi e Jay M. Gambetta. ≪Error Mitigation for Short-Depth Quantum Circuits≫. In: *Physical Review Letters* 119.18 (nov. 2017). DOI: 10.1103/physrevlett.119.180509.

[27] Tudor Giurgica-Tiron et al. ≪Digital zero noise extrapolation for quantum error mitigation≫. In: *2020 IEEE International Conference on Quantum Computing and Engineering (QCE)*. IEEE, ott. 2020. DOI: 10.1109/qce49297.2020.00045.

[28] Ewout van den Berg et al. *Probabilistic error cancellation with sparse Pauli-Lindblad models on noisy quantum processors*. 2022. arXiv: 2201.09866 [quant-ph].

[29] Ryan LaRose et al. ≪Mitiq: A software package for error mitigation on noisy quantum computers≫. In: *Quantum* 6 (ago. 2022), p. 774. DOI: 10.22331/q-2022-08-11-774.

[30] Claude Cohen-Tannoudji, Bernard Diu e Franck Laloë. *Quantum Mechanics, Volume I: Basic Concepts, Tools and Application*. 2ª ed. Wiley-VCH, 2020.

[31] Lev D. Landau e Evgenij M. Lifšits. *Fisica teorica III, Meccanica Quantistica non relativistica*. Editori Riuniti university press, 1976.

[32] Abhinav Kandala et al. ≪Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets≫. In: *Nature* 549 (set. 2017), pp. 242–246. DOI: 10.1038/nature23879.

[33] Stefano Barison et al. ≪Variational dynamics as a ground-state problem on a quantum computer≫. In: *Physical Review Research* 4.4 (dic. 2022). DOI: 10.1103/physrevresearch.4.043161.

[34] Sergey Bravyi et al. ≪Mitigating measurement errors in multiqubit experiments≫. In: *Physical Review A* 103.4 (apr. 2021). DOI: 10.1103/physreva.103.042605.

[35] Michael R Geller e Mingyu Sun. ≪Toward efficient correction of multiqubit measurement errors: pair correlation method≫. In: *Quantum Science and Technology* 6.2 (feb. 2021), p. 025009. DOI: 10.1088/2058-9565/abd5c9.

[36] Filip B. Maciejewski, Zoltán Zimborás e Michał Oszmaniec. ≪Mitigation of readout noise in near-term quantum devices by classical post-processing based on detector tomography≫. In: *Quantum* 4 (apr. 2020), p. 257. ISSN: 2521-327X. DOI: 10.22331/q-2020-04-24-257.

[37] K. Mitarai et al. ≪Quantum circuit learning≫. In: *Physical Review A* 98.3 (set. 2018). DOI: 10.1103/physreva.98.032309.

[38] Maria Schuld et al. ≪Evaluating analytic gradients on quantum hardware≫. In: *Physical Review A* 99.3 (mar. 2019). DOI: 10.1103/physreva.99.032331.

[39] Diederik P. Kingma e Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[40] James C. Spall. ≪An Overview of the Simultaneous Perturbation Method for Efficient Optimization≫. In: *Johns Hopkins Apl Technical Digest* 19 (1998), pp. 482–492.

[41] Seth Lloyd. ≪Universal Quantum Simulators≫. In: *Science* (ago. 1996). DOI: 10.1126/science.273.5278.1073.

[42] Xiao Yuan et al. ≪Theory of variational quantum simulation≫. In: *Quantum* 3 (ott. 2019), p. 191. DOI: 10.22331/q-2019-10-07-191.

[43] M. Cerezo et al. ≪Cost function dependent barren plateaus in shallow parametrized quantum circuits≫. In: *Nature Communications* 12.1 (mar. 2021). DOI: 10.1038/s41467-021-21728-w.

[44] *GitHub repository: qismib/sp-VQD*. URL: %5Curl%7Bhttps://github.com/qismib/sp-VQD%7D.

[45] Andreas Elben et al. ≪The randomized measurement toolbox≫. In: *Nature Reviews Physics* 5.1 (dic. 2022), pp. 9–24. DOI: 10.1038/s42254-022-00535-2.

[46] Hsin-Yuan Huang, Richard Kueng e John Preskill. ≪Predicting many properties of a quantum system from very few measurements≫. In: *Nature Physics* 16.10 (giu. 2020), pp. 1050–1057. DOI: 10.1038/s41567-020-0932-7.

[47] Dorit Aharonov, Vaughan Jones e Zeph Landau. *A Polynomial Quantum Algorithm for Approximating the Jones Polynomial*. 2006. arXiv: quant-ph/0511096 [quant-ph].