Quantum Portfolio Optimization

Colli Simone[1] and Merenda Saverio Mattia[1]

[1] simone.colli@studenti.unipr.it
[2] saveriomattia.merenda@studenti.unipr.it

December 27, 2024

(MM: todo)

(MM: l'ideale sarebbe ricreare le figure 2 e 3 per mantenere lo stesso stile)

1 Introduzione

L'ottimizzazione del portafoglio (PO) è un'attività finanziaria di primaria importanza, con applicazioni significative in diversi contesti, come i fondi di investimento, i piani pensionistici e altre strategie di allocazione del capitale. Data una disponibilità di budget e/o un insieme di asset, l'obiettivo è individuare operazioni ottimali all'interno di un mercato che può includere un numero elevato di asset. La corretta allocazione degli asset ha un impatto diretto sulla redditività degli investimenti, consentendo di ottenere rendimenti più elevati e una migliore gestione del rischio. Data la rilevanza economica del problema, l'ottimizzazione del portafoglio rappresenta un'area strategica sia per le istituzioni finanziarie sia per gli investitori.

Limiti dei metodi classici I metodi tradizionali, come gli approcci geometrici o gli algoritmi euristici, presentano significative limitazioni, soprattutto in termini di scalabilità ed efficienza. Con l'aumentare della complessità e delle dimensioni del mercato, la risoluzione del problema diventa rapidamente intrattabile per i computer classici. Ad esempio, algoritmi come il branch-and-bound (Land and Doig, 2010), utilizzati per trovare soluzioni esatte, faticano a gestire mercati con un numero elevato di asset.

Quantum computing L'introduzione del calcolo quantistico apre nuove possibilità per affrontare i limiti dei metodi classici. Sfruttando i principi della meccanica quantistica, come la sovrapposizione e l'entanglement, i computer quantistici promettono di risolvere problemi di ottimizzazione in modo più efficiente. In particolare, i problemi di ottimizzazione quadratica, come quello del portafoglio, possono beneficiare di algoritmi quantistici in grado di trovare soluzioni quasi ottimali in tempi significativamente ridotti rispetto ai metodi tradizionali.

Computer classici vs computer quantistici I computer classici (CPU) elaborano le informazioni utilizzando i bit, che possono assumere esclusivamente due stati, 0 o 1. Questa caratteristica limita la capacità di esplorare lo spazio delle soluzioni in parallelo, costringendo i calcoli a procedere in modo sequenziale o attraverso tecniche di parallelismo limitate.

Al contrario, i computer quantistici (QPU) sfruttano i qubit, che possono trovarsi in una sovrapposizione di stati, rappresentando simultaneamente sia 0 che 1. Grazie a questa proprietà unica, i computer quantistici sono in grado di eseguire calcoli in parallelo, esplorando uno spazio di soluzioni molto più vasto rispetto ai computer classici e rendendoli particolarmente adatti per affrontare problemi complessi come quelli di ottimizzazione.

2 Costruzione del problema

(MM: parlare di come sono stati estratti i dati (dalla libreria qiskit) e mostrare un esempio ridotto in tabella)

Per lo svolgimento di questo progetto, sono stati analizzati dataset di dimensioni contenute, selezionando un massimo di n asset distinti. Per ciascun asset i, con $1 \leq i \leq n$, è stato considerato l'intervallo temporale tra il (MM: xx/xx/xxxx e il xx/xx/xxxx). Per ogni giorno t in questo intervallo ($0 \leq t \leq T$), la performance di un asset è rappresentata dal suo prezzo di chiusura p_i^t.

La prima informazione estratta da questo dataset consiste nell'elenco P dei prezzi correnti P_i degli asset considerati:

$$P_i = p_i^t. \tag{1}$$

Inoltre, per ciascun asset, il rendimento r_i^t tra i giorni $t-1$ e t può essere calcolato come:

$$r_i^t = \frac{p_i^t - p_i^{t-1}}{p_i^{t-1}}. \tag{2}$$

Grazie a questi rendimenti, è possibile definire il rendimento atteso di un asset come una stima ragionata della sua futura performance. Supponendo una distribuzione normale dei rendimenti, la media dei loro valori in ogni momento t nel set di osservazioni storiche è un buon stimatore del rendimento atteso. Pertanto, dato l'intero dataset storico, il rendimento atteso di ciascun asset μ_i è calcolato come:

$$\mu_i = E[r_i] = \frac{1}{T}\sum_{t=1}^{T} r_i^t. \tag{3}$$

Seguendo lo stesso principio, la varianza del rendimento di ciascun asset, σ_{ij}, e la covarianza tra i rendimenti di asset differenti nel corso delle serie storiche, σ_i^2, possono essere calcolate come segue:

$$\sigma_{ij} = E[(r_i - \mu_i)(r_j - \mu_j)] = \frac{1}{T-1}\sum_{t=1}^{T}(r_i^t - \mu_i)(r_j^t - \mu_j), \tag{4}$$

$$\sigma_i^2 = E[(r_i - \mu_i)^2] = \frac{1}{T-1}\sum_{t=1}^{T}(r_i^t - \mu_i)^2. \tag{5}$$

Un portafoglio è definito come un insieme di investimenti x_i (misurati come frazione del budget o del numero di unità allocate) per ciascun asset i del mercato. Pertanto, il portafoglio è composto da un vettore di numeri reali o interi con dimensioni pari al numero di asset considerati.

Una strategia ottimale di allocazione del portafoglio punta a **massimizzare** il rendimento del portafoglio $\mu^\top x$ **minimizzando** il rischio, definito come la varianza del portafoglio $x^\top \Sigma x$, la cui radice quadrata rappresenta la volatilità del portafoglio. In questo caso, μ è il vettore dei rendimenti medi per ciascun asset i calcolato con la Formula 3,

$$\mu = \begin{bmatrix} \mu_0 \\ \mu_1 \\ \vdots \\ \mu_n \end{bmatrix}, \tag{6}$$

Σ è la matrice di covarianza calcolata con le Formule 4 e 5,

$$\Sigma = \begin{bmatrix} \sigma_0^2 & \sigma_{10} & \cdots & \sigma_{n0} \\ \sigma_{01} & \sigma_1^2 & \cdots & \sigma_{n1} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{0n} & \sigma_{1n} & \cdots & \sigma_n^2 \end{bmatrix}, \tag{7}$$

e x è il vettore delle frazioni di budget allocate per ciascun asset.

L'obiettivo di trovare un portafoglio ottimale consiste quindi nel trovare il vettore x che minimizza la funzione obiettivo seguente:

$$\mathcal{L}(x) = q x^\top \Sigma x - \mu^\top x, \tag{8}$$

dove il parametro di avversione al rischio q esprime la propensione dell'investitore al rischio, i.e., un compromesso tra rischio e rendimento.

In uno scenario realistico, il budget disponibile B è fisso. Pertanto, il vincolo secondo cui la somma degli x_i deve essere pari a 1 è valido, e può essere espresso nel seguente modo:

$$B = \sum_{i=1}^{N} x_i = 1. \tag{9}$$

Di conseguenza, il problema può essere espresso come segue:

$$\min_x (q x^\top \Sigma x - \mu^\top x), \tag{10}$$

dove:

- $x \in \{0,1\}^n$ denota il vettore delle variabili decisionali binarie, che indicano quali asset selezionare e quali no, identificati con $x_i = 1$ e $x_i = 0$, rispettivamente;
- $\mu \in \mathbb{R}^n$ definisce i rendimenti attesi degli asset;
- $\Sigma \in \mathbb{R}^{n \times n}$ specifica le covarianze tra gli asset;
- $q > 0$ controlla l'avversione al rischio del decisore;
- B denota il budget, ovvero il numero di asset da selezionare tra gli n disponibili.

Per poter risolvere il problema mediante algoritmi quantistici, è necessario formularlo senza vincoli espliciti. Per questo motivo, introduciamo un termine di penalità che favorisce le soluzioni in cui il numero di asset selezionati, i.e., il numero di 1 nel vettore x, sia il più vicino possibile al budget B.

Il problema di ottimizzazione risulta quindi:

$$\min_x \left(q x^T \Sigma x - x \mu^T + (1^T x - B)^2 \right). \tag{11}$$

Questa formulazione rappresenta un Quadratic Unconstrained Binary Optimization problem (QUBO), che può essere risolto utilizzando algoritmi di ottimizzazione quantistica basati sul principio variazionale, come il Variational Quantum Eigensolver (VQE) e il Quantum Approximate Optimization Algorithm (QAOA), i quali verranno approfonditi nelle Sezioni 3.1 e 3.2, rispettivamente.

Perchè usare il Quantum Computing Nel contesto dell'ottimizzazione di portafoglio, l'analisi della complessità computazionale rivela differenze significative tra l'approccio classico e quello quantistico. Nel caso classico, il problema ricade nella classe dei problemi NP-hard, dove lo spazio delle soluzioni cresce esponenzialmente: per n variabili binarie, abbiamo 2^n possibili soluzioni, mentre per x variabili intere che variano da 0 a $x_{\max}$, lo spazio delle soluzioni diventa $(x_{\max}+1)^x$. Per gestire questa complessità, sono stati sviluppati metodi euristici che però mostrano limitazioni pratiche, risultando efficaci solo per portafogli con pochi asset.

L'approccio quantistico, invece, sfrutta fenomeni quantistici fondamentali come l'interferenza e l'entanglement per eseguire computazioni all'interno della classe di complessità BQP (Bounded-error Quantum Polynomial). Questa classe di problemi richiede un tempo polinomiale per la risoluzione su un computer quantistico, ritornando una soluzione corretta con probabilità maggiore o uguale a $\frac{2}{3}$ (Buonaiuto *et al.*, 2023).

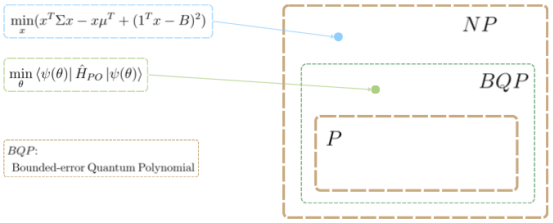


Figure 1: (MM: ricreare questa immagine)

3 Quadratic Unconstrained Binary Optimization

I problemi Quadratic Unconstrained Binary Optimization (QUBO) rappresentano una classe fondamentale di problemi di ottimizzazione in cui si cerca di minimizzare (o massimizzare) una funzione quadratica con variabili binarie, cioè variabili che possono assumere solo i valori 0 o 1. Ciò che rende i QUBO particolarmente interessanti è il fatto che molti problemi complessi di ottimizzazione possono essere riformulati in questo formato.

La caratteristica principale dei QUBO è la loro semplicità strutturale: sono problemi non vincolati, il che significa che l'unico vincolo è la natura binaria delle variabili. Nonostante questa apparente semplicità, i QUBO sono problemi NP-completi, il che significa che sono computazionalmente difficili da risolvere con metodi classici quando la dimensione del problema cresce.

L'importanza dei QUBO nel contesto del calcolo quantistico deriva dalla loro versatilità. Infatti, possiamo utilizzarli per risolvere una vasta gamma di problemi pratici come:

- la colorazione dei grafi, dove si cerca di assegnare colori a nodi di un grafo in modo che nodi adiacenti abbiano colori diversi;
- il partizionamento di numeri, dove si cerca di dividere un insieme di numeri in gruppi con somma simile;
- l'ottimizzazione di portafoglio finanziario, dove si cerca di bilanciare rischio e rendimento;

Nel contesto degli algoritmi quantistici come il VQE e il QAOA, i problemi QUBO sono particolarmente adatti perché la loro struttura si traduce naturalmente in termini di interazioni tra qubit. Questa caratteristica li rende ideali per l'implementazione su computer quantistici, dove le interazioni tra qubit possono essere controllate e manipolate per trovare soluzioni ottimali.

(MM: Il collegamento tra i problemi QUBO e gli algoritmi quantistici come VQE e QAOA è particolarmente interessante. Per utilizzare questi algoritmi quantistici, il primo passo consiste nel mappare il problema QUBO in un Hamiltoniano quantistico. Questa mappatura viene realizzata traducendo le variabili binarie classiche x_i in operatori di Pauli-Z, dove lo stato del qubit rappresenta il valore della variabile binaria.

Una volta che il problema è stato codificato nell'Hamiltoniano, possiamo utilizzare sia il VQE che il QAOA per trovare la soluzione. Mentre il VQE è un approccio più generale che permette di utilizzare diversi tipi di circuiti quantistici (ansatz), il QAOA utilizza una struttura specifica alternando due operatori: l'Hamiltoniano del problema (che codifica il QUBO) e l'Hamiltoniano di mixing che permette di esplorare lo spazio delle soluzioni.

La differenza chiave nell'utilizzo di questi due algoritmi sta nel modo in cui cercano la soluzione: - Il VQE è più flessibile nella scelta della struttura del circuito ma potrebbe richiedere più parametri da ottimizzare - Il QAOA ha una struttura più rigida ma potenzialmente più efficiente per problemi QUBO, poiché è stato specificamente progettato per problemi di ottimizzazione combinatoria

In entrambi i casi, questi algoritmi sfruttano le proprietà quantistiche come sovrapposizione e entanglement per esplorare lo spazio delle soluzioni in modo più efficiente rispetto agli algoritmi classici, rendendo possibile la risoluzione di problemi QUBO di dimensioni maggiori su hardware quantistico.

)

3.1 Variational Quantum Eigensolver

Il Variational Quantum Eigensolver (VQE) è un algoritmo ibrido che combina l'uso di computer quantistici e classici per risolvere problemi di ottimizzazione. Il suo funzionamento si basa sul principio variazionale e mira a trovare lo stato di energia minima di un sistema quantistico.

L'idea principale è quella di parametrizzare un circuito quantistico attraverso un insieme di parametri θ e utilizzare questi parametri per minimizzare l'energia del sistema, definita come:

$$E(\theta) = \frac{\langle\psi(\theta)|H|\psi(\theta)\rangle}{\langle\psi(\theta)|\psi(\theta)\rangle}, \tag{12}$$

dove H è l'Hamiltoniano che descrive il nostro problema di ottimizzazione e $\psi(\theta)$ è la funzione d'onda parametrizzata. In particolare, lo stato fondamentale E_0 corrisponde allo stato fondamentale (*ground state*) di energia minima:

$$E_0 \leq \frac{\langle\psi(\theta)|H|\psi(\theta)\rangle}{\langle\psi(\theta)|\psi(\theta)\rangle}. \tag{13}$$

Quindi, il compito del VQE è trovare l'insieme ottimale di parametri, tale che l'energia associata allo stato sia quasi indistinguibile dal suo stato fondamentale, cioè trovare l'insieme di parametri θ, corrispondente all'energia $E_{\min}$, per il quale $|E_{\min} - E_0| < \epsilon$, dove ϵ è una costante arbitrariamente piccola. Questo problema può essere formulato su un computer quantistico come una serie di gates, che vengono applicate allo stato iniziale per realizzare un ansatz strutturato per il problema Hamiltoniano.

Convenzionalmente, lo stato iniziale è impostato per essere lo stato di vuoto, i.e., per un sistema di N qubit $|0\rangle^{\otimes N} = |0\rangle$. Quindi, il problema di minimizzare l'energia del sistema (Equazione 12) può essere espresso come:

$$E_{\min} = \min_{\theta} \langle 0|U^{\dagger}(\theta) H U(\theta)|0\rangle, \tag{14}$$

dove $U(\theta)$ è l'operatore unitario parametrizzato che fornisce la funzione d'onda ansatz quando applicato allo stato iniziale, e $E_{\min}$ è l'energia associata all'ansatz parametrizzato.

Per essere efficientemente implementato in un circuito quantistico, è importante che H venga espresso nella forma:

$$H = \sum_{l} c_l P_l = \sum_{l} c_l \otimes \sigma_j^i, \tag{15}$$

dove P_l sono stringhe di Pauli rappresentate dal prodotto tensore degli operatori di Pauli $\sigma_j^i \in \{I, \sigma_X, \sigma_Y, \sigma_Z\}$, con j che denota il qubit su cui l'operatore agisce e i il termine dell'Hamiltoniano. I coefficienti c_l sono pesi reali o complessi, adeguati per definire l'Hamiltoniano specifico del problema. In questa rappresentazione, l'Hamiltoniano è espresso come una combinazione lineare di stringhe di Pauli.

Quindi, l'obiettivo del VQE è risolvere il seguente problema di ottimizzazione:

$$E_{\min} = \min_{\theta} \sum_{l} c_l \langle 0|U^{\dagger}(\theta) P_l U(\theta)|0\rangle. \tag{16}$$

In questo contesto, si cerca lo stato $|\psi(\theta)\rangle$ che corrisponde allo stato fondamentale di H.

Il calcolo del valore atteso di una stringa di Pauli P_l è essenziale. Questo valore è ottenuto misurando ogni qubit coinvolto nella stringa P_l, operando nella base di misura adeguata. Ad esempio, per uno stato generico del tipo $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, il valore atteso sull'operatore di Pauli σ_Z è dato da:

$$\langle \psi|\sigma_Z|\psi\rangle = |\alpha|^2 - |\beta|^2. \tag{17}$$

Questo valore rappresenta la probabilità di osservare lo stato $|0\rangle$ meno la probabilità di osservare lo stato $|1\rangle$. Misure relative a σ_X o σ_Y richiedono una rotazione preliminare nella base di misura σ_Z.

Dato che i risultati delle misurazioni quantistiche sono binari, è necessario ripetere l'esperimento più volte per approssimare al meglio il valore medio di ogni termine. Questo processo è ripetuto separatamente per ogni stringa P_l che compone l'Hamiltoniano.

L'approccio VQE mira a bilanciare la complessità del circuito quantistico con il numero di misurazioni richieste, permettendo un'ottimizzazione efficiente degli stati parametrizzati. Il processo di ottimizzazione avviene in modo iterativo. Inizialmente, il computer quantistico prepara uno stato quantistico utilizzando i parametri correnti. Successivamente, viene misurata l'energia di questo stato preparato. Sulla base di questa misura, un ottimizzatore classico aggiorna i parametri nel tentativo di minimizzare l'energia del sistema. Questo ciclo di preparazione, misurazione e aggiornamento viene ripetuto fino a raggiungere la convergenza, ovvero fino a quando l'energia non può essere ulteriormente minimizzata in modo significativo (Figura 2).

Questo approccio ibrido sfrutta i punti di forza di entrambe le tipologie di computer: il computer quantistico si occupa della preparazione e della misurazione degli stati quantistici, mentre il computer classico gestisce l'ottimizzazione dei parametri. Nel contesto dell'ottimizzazione di portafoglio, il VQE viene impiegato per trovare la configurazione ottimale degli asset che minimizza una funzione obiettivo, tenendo conto sia del rischio che del rendimento atteso.

3.2 Quantum Approximate Optimization Algorithm

Il Quantum Approximate Optimization Algorithm (QAOA) è un algoritmo variazionale quantistico progettato per trovare soluzioni approssimate a problemi di ottimizzazione combinatoria. Il suo

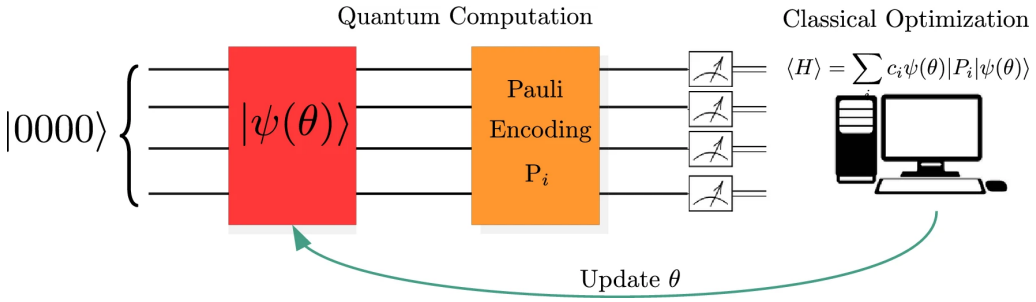


Figure 2: Schema del funzionamento del Variational Quantum Eigensolver (Buonaiuto *et al.*, 2023).

funzionamento si basa sull'alternanza di due operatori: un operatore di *costo* e un operatore di *mixing*, il quale quest'ultimo ha il compito di esplorare lo spazio delle soluzioni.

L'algoritmo opera attraverso una sequenza di strati (*layers*) quantistici, dove ogni strato è composto da due parti principali:

$$\hat{U}_C(\gamma) = e^{-i\gamma\hat{H}_C} \quad \text{(operatore di costo)}, \tag{18}$$

$$\hat{U}_M(\beta) = e^{-i\beta\hat{H}_M} \quad \text{(operatore di mixing)}, \tag{19}$$

dove $\hat{H}_C$ è l'Hamiltoniano di costo che codifica il problema da ottimizzare, e $\hat{H}_M$ è l'Hamiltoniano di mixing che permette di esplorare lo spazio delle soluzioni. I parametri γ e β sono parametri variazionali che vengono ottimizzati durante l'esecuzione dell'algoritmo.

Il circuito quantistico inizia preparando tutti i qubit nello stato di sovrapposizione:

$$|s\rangle = |+\rangle^{\otimes n} = \frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} |x\rangle. \tag{20}$$

Lo stato finale dopo p layers è dato da:

$$|\psi_p(\gamma,\beta)\rangle = e^{-i\beta_p\hat{H}_M} e^{-i\gamma_p\hat{H}_C} \cdots e^{-i\beta_1\hat{H}_M} e^{-i\gamma_1\hat{H}_C} |s\rangle \tag{21}$$

L'obiettivo è massimizzare il valore atteso dell'Hamiltoniano di costo:

$$F_p(\gamma,\beta) = \langle\psi_p(\gamma,\beta)|\hat{H}_C|\psi_p(\gamma,\beta)\rangle \tag{22}$$

Il processo di ottimizzazione avviene in modo ibrido: il circuito quantistico prepara e misura gli stati, mentre un ottimizzatore classico aggiorna i parametri γ e β per massimizzare F_p. Questo processo viene ripetuto fino a quando non si trova una soluzione soddisfacente al problema di ottimizzazione (Figura 3).

L'algoritmo QAOA rappresenta quindi un ponte tra il calcolo quantistico e quello classico, sfruttando i vantaggi di entrambi i paradigmi per risolvere problemi di ottimizzazione complessi. La sua efficacia dipende dal numero di layer p utilizzati e dalla qualità dell'ottimizzazione dei parametri.

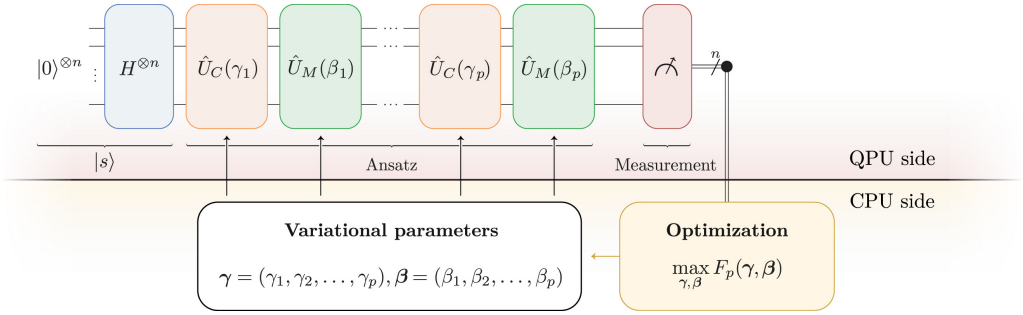


Figure 3: Schema del funzionamento del Quantum Approximate Optimization Algorithm (Blekos *et al.*, 2024).

4 Risoluzione del problema

Per simulare il problema di ottimizzazione del portafoglio, abbiamo utilizzato un approccio basato su un modello generativo per produrre dati storici. La generazione dei dati è stata realizzata tramite una classe fornita da Qiskit, `RandomDataProvider`, che, a partire da parametri come il numero di asset, il periodo temporale e un seme per la riproducibilità, ha prodotto una tabella di valori rappresentante l'andamento dei prezzi degli asset nel tempo (Tabella I). Inoltre, per rendere più intuitiva la visualizzazione dei dati, abbiamo sviluppato uno script che crea un grafico per ciascun asset, rappresentando l'andamento temporale dei prezzi (Figura 4a).

Data	**Ticker_0**	**Ticker_1**	**Ticker_2**	**Ticker_3**
2016–01–01	80.0573	38.8648	69.2682	76.1653
2016–01–02	80.3390	38.7966	69.5354	76.6583
2016–01–03	79.4722	39.0100	70.7954	76.0500
...	...	...	...	...

Table I: Esempio di dati generati dalla classe `RandomDataProvider` con un numero di asset uguale a 4 e un periodo temporale che parte dal 2016–01–01.

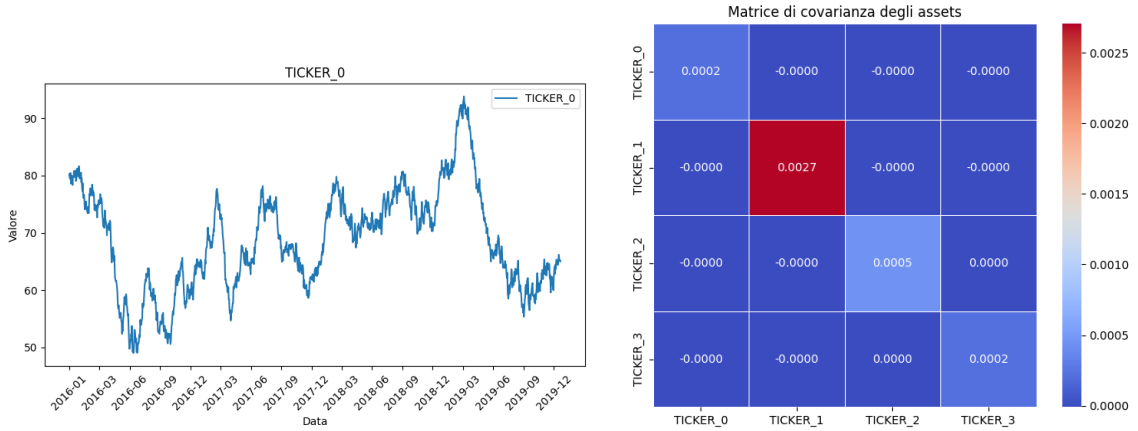
Successivamente, sono stati calcolati i rendimenti medi per ciascun asset (Formula 6) e la matrice di covarianza (Formula 7) utilizzando i metodi offerti dalla classe `RandomDataProvider`. Questi due elementi rappresentano una componente fondamentale per l'ottimizzazione del portafoglio, in quanto i rendimenti medi forniscono una misura dell'aspettativa di guadagno, mentre la matrice di covarianza descrive la relazione tra le variazioni dei prezzi dei diversi asset.

I calcoli sono stati effettuati mediante i seguenti metodi:

```
mu = data_provider.get_period_return_mean_vector()
sigma = data_provider.get_period_return_covariance_matrix()
```

Listing 1: Calcolo dei rendimenti attesi e della matrice di covarianza.

Per favorire una migliore comprensione e visualizzazione della matrice di covarianza, ne è stato generato un grafico (Figura 4b): questo consente di evidenziare visivamente le correlazioni tra i diversi asset.



(a) Andamento temporale di un asset.

(b) Esempio di matrice di covarianza.

Figure 4: Visualizzazione (a) dell'andamento dei prezzi degli asset generati, e (b) della matrice di covarianza calcolata.

Dopo aver calcolato i rendimenti medi e la matrice di covarianza per ciascun asset, procediamo con la configurazione del problema di ottimizzazione. Il fattore di rischio viene impostato a *0.2*, permettendo di bilanciare il trade-off tra rendimento e rischio del portafoglio. Il budget viene definito come i due terzi del numero totale di asset (`assets / 3 * 2`). Viene inoltre definita la penalità (`penalty`) uguale al valore del budget per gestire la violazione dei vincoli.

Il problema viene quindi configurato attraverso l'inizializzazione di un'istanza della classe `PortfolioOptimization`, alla quale vengono passati i parametri fondamentali: il vettore dei rendimenti attesi (`expected_returns`), la matrice di covarianza (`covariances`), il fattore di rischio e il budget. La configurazione finale produce un output che specifica il numero totale di asset disponibili, il budget allocato, la penalità impostata e il fattore di rischio utilizzato. Il problema viene infine convertito in un programma quadratico attraverso il metodo `to_quadratic_program` per la successiva ottimizzazione.

```
risk_factor = 0.2
budget = int(assets / 3 * 2)
penalty = budget

po = PortfolioOptimization(
    expected_returns=mu,
    covariances=sigma,
    risk_factor=risk_factor,
    budget=budget
)
qp = po.to_quadratic_program()
```

Listing 2: Configurazione del problema di ottimizzazione del portafoglio.

Risoluzione classica del problema Per ottenere una soluzione di riferimento al nostro problema di ottimizzazione, utilizziamo un approccio classico basato su autovalori. In particolare, impieghiamo il risolutore `NumPyMinimumEigensolver`, un algoritmo che calcola in modo esatto gli autovalori del problema. Questo risolutore viene integrato attraverso `MinimumEigenOptimizer`, un wrapper che fornisce un'interfaccia conveniente per utilizzare il solver e risolvere il problema di ottimizzazione.

```
exact_mes = NumPyMinimumEigensolver()
exact_eigensolver = MinimumEigenOptimizer(exact_mes)
result = exact_eigensolver.solve(qp)
```

Listing 3: Risoluzione classica del problema utilizzando `NumPyMinimumEigensolver`.

Questa implementazione classica serve come benchmark fondamentale per valutare le prestazioni dei metodi quantistici che verranno implementati successivamente.

Si può osservare che nel risultato ottenuto nella Figura 9c, il risolutore classico fornisce un unico risultato, il quale corrisponde a quello ottimale, con una probabilità del 100%. Come già discusso nella Sezione 2, questo metodo è rapido quando tratta un numero ridotto di asset, ma diventa sempre più lento man mano che il numero di quest'ultimi aumenta. In definitiva, come già accennato, l'obiettivo principale del quantum computing non è tanto migliorare il risultato ottenuto, quanto piuttosto accelerare la risoluzione dei problemi.

Risoluzione con VQE Per risolvere il problema del PO con un approccio ibrido quantistico-classico, abbiamo scelto, come prima soluzione, di implementare il VQE. Come già spiegato nella Sezione 3.1, questo algoritmo combina la potenza computazionale dei circuiti quantistici con l'efficienza degli ottimizzatori classici, utilizzando un ansatz parametrizzato per esplorare lo spazio degli stati quantistici e individuare la configurazione che minimizza l'energia del sistema.

L'ansatz scelto è il circuito `TwoLocal`, caratterizzato da porte R_y per la rotazione dei qubit, porte CZ per l'entanglement, e l'inserimento di barriere per favorire la leggibilità:

```
ry = TwoLocal(assets, "ry", "cz",
              reps=1,
              entanglement="full",
              insert_barriers=True)
```

Listing 4: (MM: todo)

Il circuito parametrizzato è stato integrato nel metodo `SamplingVQE`, che utilizza un campionatore (`Sampler()`) e l'ottimizzatore COBYLA per la minimizzazione dei parametri. La soluzione del problema è stata ottenuta tramite `MinimumEigenOptimizer`, come mostrato nel seguente codice:

```
svqe_mes = SamplingVQE(sampler=Sampler(),
                       ansatz=ry,
                       optimizer=cobyla)
svqe = MinimumEigenOptimizer(svqe_mes, penalty)
result_svqe = svqe.solve(qp)
```

Listing 5: (MM: todo)

La funzione `print_result` è stata utilizzata per visualizzare la soluzione ottimale, includendo la configurazione degli asset selezionati e il valore della funzione obiettivo.

La Figura 5 mostra lo schema del circuito generato dall'ansatz `TwoLocal`, evidenziando le porte R_y, le connessioni entangled e le barriere di separazione.

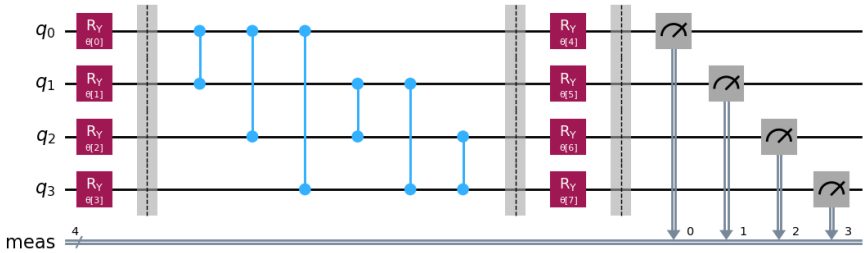


Figure 5: Circuito parametrizzato per il VQE, generato con l'ansatz `TwoLocal`.

Un esempio di risultato di un'esecuzione del VQE è mostrato nella Figura 9a, in cui si evidenziano la configurazione ottimale degli asset e il valore della funzione obiettivo. Questo risultato rappresenta l'allocazione di portafoglio ottimale secondo il modello definito.

Risoluzione con QAOA Un secondo approccio che abbiamo deciso di utilizzare per risolvere il problema del PO è stato quello del QAOA. Come descritto nella Sezione 3.2, questo algoritmo utilizza un'architettura parametrizzata che integra circuiti quantistici e ottimizzatori classici, con l'obiettivo di approssimare la configurazione ottimale attraverso un processo iterativo di adattamento dei parametri.

In una prima fase, abbiamo provato a implementare manualmente il QAOA, ma questa soluzione si è rivelata poco efficiente dal punto di vista computazionale, richiedendo tempi di esecuzione eccessivamente lunghi. Per una questione di ottimizzazione e scalabilità, è stato quindi deciso di utilizzare la classe `QAOA` offerta da Qiskit, che fornisce un'implementazione già ottimizzata dell'algoritmo.

Il circuito parametrizzato del QAOA è stato configurato con il numero di ripetizioni (`reps=1`) e l'ottimizzatore COBYLA per la ricerca dei parametri ottimali. Il codice per configurare e risolvere il problema è il seguente:

```
qaoa_mes = QAOA(sampler=Sampler(), optimizer=cobyla, reps=1)
qaoa = MinimumEigenOptimizer(qaoa_mes, penalty)
result_qaoa = qaoa.solve(qp)
```

Listing 6: (MM: todo)

Il circuito generato rappresenta il cuore del processo di ottimizzazione, poiché definisce lo spazio di soluzioni esplorato dall'algoritmo. La Figura 6 mostra lo schema del circuito generato, evidenziando i layer parametrizzati e le connessioni tra i qubit.

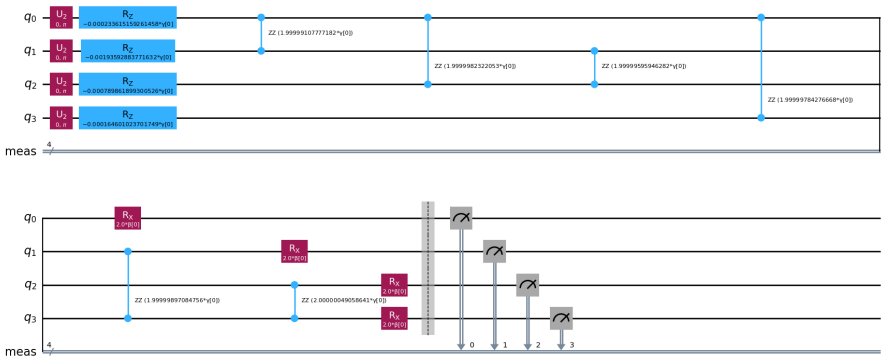


Figure 6: Circuito parametrizzato per il QAOA.

Un esempio di risultato di un'esecuzione del QAOA è mostrato nella Figura 9b, che evidenzia la configurazione ottimale degli asset e il valore della funzione obiettivo.

4.1 Introduzione del rumore nell'implementazione degli algoritmi

Finora, gli algoritmi VQE e QAOA sono stati simulati assumendo un sistema ideale, privo di rumore. Tuttavia, per avvicinare l'implementazione alla realtà dei computer quantistici attuali, abbiamo introdotto un modello di rumore basato su un backend simulato, `FakeTorino`, offerto da Qiskit. Questo backend emula le caratteristiche e le imperfezioni di un dispositivo quantistico reale, come errori di decoerenza e di misurazione.

(MM: scrivere le percentuali dei rumori introdotti)

Il modello di rumore è stato generato utilizzando la classe `NoiseModel` e applicato al simulatore quantistico `AerSimulator`. Il codice utilizzato per configurare il simulatore rumoroso è il seguente:

```
backend = FakeTorino()
noise_model = NoiseModel.from_backend(backend)
simulator = AerSimulator(noise_model=noise_model)
```

Listing 7: (MM: todo)

Implementazione del VQE con rumore Per il VQE, è stato utilizzato un circuito `TwoLocal` come ansatz. Il circuito è stato (MM: transpiled) per adattarlo alle caratteristiche del simulatore rumoroso. Successivamente, il `SamplingVQE` è stato configurato con l'ansatz transpiled e l'ottimizzatore COBYLA, per trovare i parametri ottimali minimizzando l'energia:

```
ansatz = TwoLocal(
    num_qubits=assets,
    rotation_blocks="ry",
    entanglement_blocks="cz",
    reps=1,
    entanglement="full",
    insert_barriers=True
)
transpiled_ansatz = transpile(ansatz, simulator)
svqe_mes_noisy = SamplingVQE(sampler=Sampler(),
                             ansatz=transpiled_ansatz,
                             optimizer=cobyla)
svqe_noisy = MinimumEigenOptimizer(svqe_mes_noisy, penalty)
result_svqe_noisy = svqe_noisy.solve(qp)
```

Listing 8: (MM: todo)

La Figura 7 mostra il circuito transpiled utilizzato per il VQE con rumore.

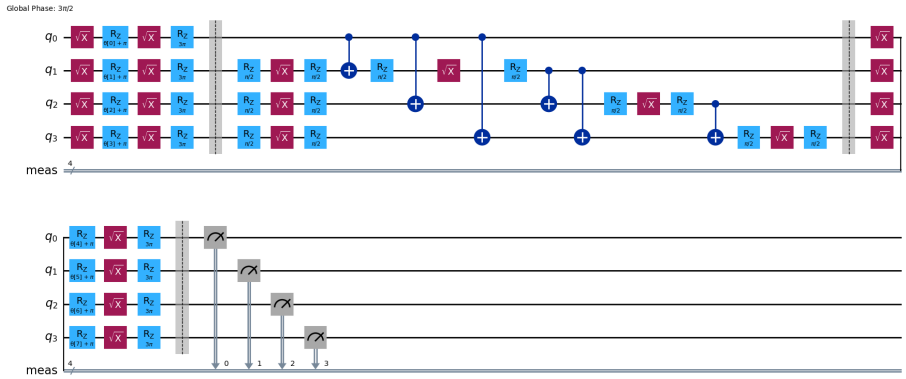


Figure 7: Circuito parametrizzato per il VQE con rumore, generato con l'ansatz `TwoLocal`.

Implementazione del QAOA con rumore Anche per il QAOA è stato adottato un approccio simile. L'ansatz del QAOA è stato decomposto e (MM: transpiled) per adattarsi al simulatore rumoroso. La classe `QAOA` è stata utilizzata con l'ottimizzatore COBYLA e il numero di ripetizioni impostato a `reps=1`:

```
    qaoa_mes_noisy = QAOA(sampler=Sampler(), optimizer=cobyla, reps=1)
    qaoa_noisy = MinimumEigenOptimizer(qaoa_mes_noisy, penalty)
    _ = qaoa_noisy.solve(qp)

    decomposed_circuit_noisy = qaoa_mes_noisy.ansatz.decompose()
    transpiled_circuit_noisy = transpile(decomposed_circuit_noisy, simulator)
```

```
    qaoa_mes_noisy.ansatz = transpiled_circuit_noisy
    qaoa_noisy = MinimumEigenOptimizer(qaoa_mes_noisy, penalty)
    result_decomposed = qaoa_noisy.solve(qp)
```

Listing 9: (MM: fix questo codice (è sbagliato))

(MM: In questa implementazione, è stato necessario eseguire il metodo `solve` due volte. Questo perché Qiskit genera in automatico il circuito quantistico in base al tipo di problema che gli viene fornito. La prima esecuzione di `solve` viene utilizzata per generare il circuito, mentre la seconda esecuzione viene effettuata dopo aver adattato il circuito al simulatore rumoroso.

La Figura 8 rappresenta il circuito transpiled per il QAOA, mentre i risultati, comprensivi della configurazione degli asset e della funzione obiettivo, sono stati stampati tramite `print_result`.)

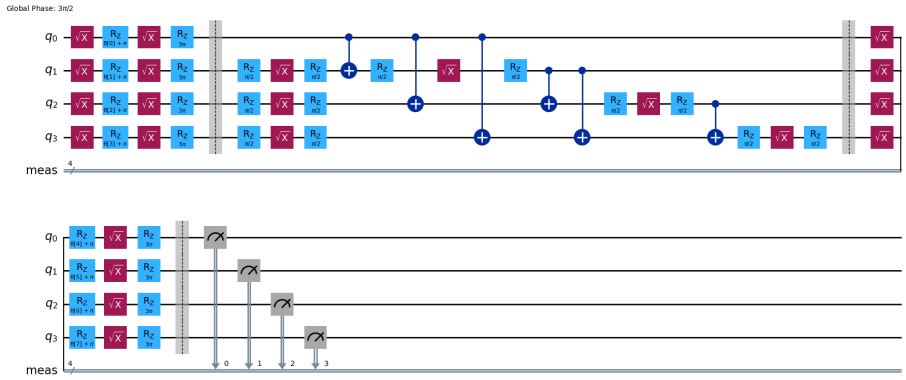


Figure 8: Circuito parametrizzato per il QAOA con rumore. (MM: todo: mettere il circuito giusto (questo è il VQE noisy))

5 Risultati

Qualifica	**# base**	**# effettivo**
Docenza di riferimento	9	$\lfloor 9 \times (1+w) \rfloor$
Docenti a tempo indeterminato	5	$\lfloor 5 \times (1+w) \rfloor$
Docenti a contratto	2	$\lfloor 2 \times (1+w) \rfloor$

Table II: Formule per il calcolo del numero di docenti di riferimento in base alla numerosità degli studenti.

```
    hello world
```

Listing 10: Esempio

Acknowledgments

```
Optimal: selection [1. 0. 1. 0.], value 0.0010

----------------- Full result --------------------
selection   value       probability
-----------------------------------------------------
 [1 0 1 0]  0.0010      0.5254
 [0 1 1 0]  0.0027      0.4658
 [0 1 0 0]  0.0019      0.0029
 [0 0 1 0]  0.0008      0.0020
 [1 0 0 0]  0.0002      0.0020
 [1 1 0 0]  0.0022      0.0010
 [0 0 0 0]  0.0000      0.0010
```

(a) Risultato ottenuto con VQE.

```
Optimal: selection [1. 0. 0. 1.], value 0.0004

----------------- Full result --------------------
selection   value       probability
-----------------------------------------------------
 [0 1 0 1]  0.0021      0.1494
 [1 0 0 1]  0.0004      0.1309
 [1 0 1 0]  0.0010      0.1250
 [1 1 0 0]  0.0022      0.1172
 [0 0 1 1]  0.0010      0.1123
 [0 1 1 0]  0.0027      0.1104
 [1 1 0 1]  0.0023      0.0352
 [1 0 1 1]  0.0012      0.0332
 [0 0 0 1]  0.0002      0.0312
 [1 0 0 0]  0.0002      0.0303
 [1 1 1 0]  0.0030      0.0293
 [0 1 0 0]  0.0019      0.0264
 [0 0 1 0]  0.0008      0.0254
 [0 1 1 1]  0.0029      0.0225
 [0 0 0 0]  0.0000      0.0117
 [1 1 1 1]  0.0031      0.0098
```

(b) Risultato ottenuto con QAOA.

```
Optimal: selection [0. 1. 0. 1.], value -0.0002

----------------- Full result --------------------
selection   value       probability
-----------------------------------------------------
-[0 1 0 1]  -0.0002     1.0000
```

(c) Risultato ottenuto con il risolutore classico.

Figure 9: Risultati ottenuti con (a) VQE, (b) QAOA e (c) risolutore classico.

References

Blekos, Kostas *et al.*, (2024). "A review on quantum approximate optimization algorithm and its variants", *Physics Reports*, Vol. 1068, pp. 1–66.

Buonaiuto, Giuseppe *et al.*, (2023). "Best practices for portfolio optimization by quantum computing, experimented on real quantum devices", *Scientific Reports*, Vol. 13 No. 1, p. 19434.

Land, Ailsa H and Doig, Alison G (2010). *An automatic method for solving discrete programming problems*, Springer.