Rješavanje svojstvenog problema na kvantnim računalima korištenjem varijacijske metode

Javorčić, Karlo

Sveučilište u Splitu
Prirodoslovno – matematički fakultet

Rješavanje svojstvenog problema na kvantnim računalima korištenjem varijacijske metode

Diplomski rad

Karlo Javorčić

Split, rujan 2023.

Temeljna dokumentacijska kartica

Sveučilište u Splitu
Prirodoslovno – matematički fakultet
Odjel za fiziku
Ruđera Boškovića 33, 21000 Split, Hrvatska

Diplomski rad

Rješavanje svojstvenog problema na kvantnim računalima korištenjem varijacijske metode

Karlo Javorčić

Sveučilišni diplomski studij Fizika, Računarska fizika

Sažetak:

U radu smo prvo prikazali osnove kvantnog računanja, a zatim pomoću tih osnova izgradili jednostavne kvantne algoritme, poput kvantne teleportacije. Zatim smo objasnili VQE algoritam koji rješava svojstveni problem na kvantnim računalima pomoću varijacijske metode. Razmatrane smo algoritme implementirali korištenjem Qiskita, softverskog računalnog okvira razvijenog od strane IBM-a, koji omogućuje kreiranje kvantnih programa i njihovo izvođenje na IBM-ovim kvantnim uređajima ili lokalnim simulatorima. U ovom je radu korišteno više simulatora. Primijenili smo VQE algoritam koristeći Qiskit na jednostavni testni hamiltonijan, molekulu vodika i molekulu litijeva hidrida. Prilikom izvođenja mijenjali smo razne atribute VQE algoritma i uspoređivali kako pojedine promjene utječu na energije osnovnog stanja dobivene VQE algoritmom. Za molekule smo također napravili analizu ovisnosti energije osnovnog stanja o udaljenosti između atoma u molekuli te smo provjeravali je li postignuta kemijska preciznost. Kroz ove primjere smo pokazali osnove implementacije VQE-a, neke od potencijalnih prepreka VQE-a te raspravili mogućnosti poboljšanja u odabranim primjerima.

Ključne riječi: kvantno računanje, kvantni algoritmi, Qiskit, VQE, optimizator, ansatz, energija osnovnog stanja

Rad sadrži: 71 stranicu, 26 slika, 13 tablica, 39 literaturnih navoda. Izvornik je na hrvatskom jeziku.

Mentor: prof. dr. sc. Leandra Vranješ Markić

Ocjenjivači: prof. dr. sc. Leandra Vranješ Markić,
izv. prof. dr. sc. Petar Stipanović,
Josipa Šćurla, mag. phys.

Rad prihvaćen: 19. 9. 2023.

Rad je pohranjen u Knjižnici Prirodoslovno – matematičkog fakulteta, Sveučilišta u Splitu.

Basic documentation card

University of Split
Faculty of Science
Department of Physics
Ruđera Boškovića 33, 21000 Split, Croatia

Master thesis

Variational Quantum Eigensolver

Karlo Javorčić

University graduate study programme Physics, Computational Physics

Abstract:
In this thesis, we first introduced the fundamentals of quantum computing and then built simple quantum algorithms like quantum teleportation. We then delved into the basics of the VQE algorithm, which solves the eigenvalue problem on quantum computers using the variational method. We used Qiskit, a software framework developed by IBM, to create and execute quantum programs on IBM's quantum devices and local simulators. In this thesis, we also used various types of simulators. We applied the VQE algorithm using Qiskit to a simple test Hamiltonian, a hydrogen molecule and a lithium hydride molecule. During the execution, we modified various attributes of the VQE algorithm and compared how these changes affect the ground state energies obtained. We also analyzed the dependency of the ground state energy on the distance between the atoms in the molecules and checked for chemical accuracy. Through these examples, we demonstrated the basics of implementing the VQE, highlighted some potential challenges and suggested areas for improvement in these given examples.

Keywords: quantum computing, quantum algorithms, Qiskit, VQE, optimizers, ansatz, ground state energy.

Thesis consists of: 71 pages, 26 figures, 13 tables, 39 references. Original language: Croatian.

Supervisor: Prof. Dr. Leandra Vranješ Markić

Reviewers: Prof. Dr. Leandra Vranješ Markić,
Asoc. Prof. Dr. Petar Stipanović,
Josipa Šćurla, M. Phys

Thesis accepted: Sep 19, 2023

Thesis is deposited in the library of the Faculty of Science, University of Split.

Zahvaljujem svojoj mentorici, prof. dr. sc. Leandri Vranješ Markić na vodstvu, pristupačnosti, ljubaznosti i savjetima tijekom pisanja ovoga rada.
Hvala mojoj obitelji, prijateljima i Nikolini na neizmjernoj podršci i ljubavi tijekom svih ovih godina. Hvala što ste bili uz mene u svim lijepim, a pogotovo teškim trenutcima. Uz vas je sve bilo ljepše i lakše.
Najveću zahvalnost dugujem svojim roditeljima, bez vas ništa ovo ne bi bilo moguće. Hvala vam što postojite.

Sadržaj

1 Uvod

Kvantna računalna revolucija predstavlja prekretnicu u načinu na koji razmišljamo o računalima i njihovim mogućnostima. Dugo su polja kvantne fizike i računalnih znanosti bila potpuno odvojena. Kvantna fizika je nastala kao rezultat nedostataka klasične fizike, dok su se računalne znanosti razvile kako bi zamijenile ljude u složenim izračunima. Međutim, krajem 20. stoljeća, fizičari su počeli primjećivati da se računalna teorija može spojiti s principima kvantne mehanike, što je dovelo do brzog razvoja područja kvantne teorije informacija i početka razvoja ideje o kvantnim računalima. Među prvima koji je ovo primijetio bio je Richard Feyman koji postulirao da za simulaciju kvantnih sustava treba izgraditi kvantna računala [1].

Tri principa iz kvantne fizike su se pokazala iznimno važnim za područje kvantnih računala, a to su: princip superpozicije, kvantna spregnutost i reverzibilnost kvantnih sustava. U klasičnom računalu, bit može biti ili u stanju 0 ili u stanju 1. Međutim, qubit, osnovna jedinica informacija u kvantnom računalu, može biti u stanju koje je kombinacija oba, zahvaljujući superpoziciji. To omogućuje kvantnom računalu da obrađuje mnogo više informacija odjednom nego što to može klasično računalo. Kvantno sprezanje omogućuje kvantnim računalima da izvode složene operacije koje povezuju više qubita, što je ključno za mnoge kvantne algoritme. Kvantna računala koriste reverzibilne operacije, što znači da se svaka operacija može "poništiti" ili vratiti natrag. Ovo je u suprotnosti s klasičnim računalima koja često koriste nepovratne operacije.

David Deutsch, fizičar s Oxforda, dao je detaljan opis kako bi kvantna računala trebala funkcionirati te je razvio algoritam koji bi se brže izvršavao na kvantnim računalima [2]. Ovo je izuzetno važno jer se u računalnim znanostima ocjenjuje učinkovitost algoritama. Tu se prvi put pokazalo da kvantna računala u određenim segmentima mogu biti nadmoćnija od klasičnih računala. Brojni fizičari su potom dalje razvijali polje, a David DiVincenzo formalizirao je kriterije za ono što zapravo predstavlja kvantno računalo [2]. U dobu smo kada je izgradnja hardvera za kvantna računala tek u začetku. Iako se suočavamo s brojnim izazovima, velike korporacije kao što su IBM, Google i mnoge druge, ali i nacionalne vlade, ulažu značajne resurse u razvoj kvantne tehnologije. Njihova ulaganja svjedoče o ogromnom potencijalu ovog područja.

U kontekstu ovog rada, poseban naglasak stavljen je na rješavanje svojstvenog problema na kvantnim računalima korištenjem varijacijske metode (engl. Variational Quantum Eigensolver (VQE)). VQE predstavlja jedan od najperspektivnijih kvantnih algoritama, posebno dizajniran za iskorištavanje mogućnosti današnjih kvantnih računala koja su još uvijek podložna pogreškama, poznatih kao "Noisy Intermediate-Scale Quantum" (NISQ) računala. Za razliku od nekih drugih kvantnih algoritama koji zahtijevaju veliku kvantnu dubinu i stotine ili čak tisuće qubita, VQE je posebno prilagođen da radi u okruženju s ograničenim resursima i visokom stopom pogrešaka, što ga čini idealnim za primjene današnjih kvantnih računala [3].

Srž VQE algoritma leži u njegovoj varijacijskoj metodi kojom se kombinirajući kvantne i klasične postupke pronalazi energija osnovnog stanja nekog kvantnog sustava. U kontekstu

kvantne kemije, to potencijalno omogućuje precizno modeliranje molekula i predviđanje njihovih svojstava s neviđenom točnošću [3].

Uzimajući u obzir brzi razvoj kvantne tehnologije i očekivanja da će NISQ era računala uskoro pokazati kvantnu nadmoć nad klasičnim računalima, VQE stoji u središtu te revolucije kao jedan od algoritama koji će najvjerojatnije prvi postići ovu prekretnicu [3].

U drugom poglavlju uvodimo osnovne pojmove kvantnog računanja, kvantne bitove i kvantna vrata. Zatim, u trećem poglavlju, opisujemo kako izgraditi jednostavan kvantni algoritam koristeći Qiskit, softverski računalni okvir razvijen od strane IBM-a. U četvrtom poglavlju razmatramo metodologiju i korake VQE algoritma, koje u petom poglavlju primjenjujemo prvo na testni hamiltonijan, a zatim na molekule vodika i litijeva hidrida. U zaključku dajemo usporedbu s literaturom i moguća unaprijeđenja provedenih izračuna.

2 Osnovni pojmovi kvantnog računanja

2.1 O kvantnim računalima

Da bismo razumjeli kvantna računala, prvo se moramo upoznati s osnovama klasičnih računala. U klasičnim računalima, informacije se prenose putem bitova. Bit može imati vrijednost 0 ili 1, a kombinacijom bitova dobivaju se različite informacije. Klasična računala koriste logičke sklopove kao osnovne komponente za obavljanje računskih operacija. Jedan primjer takvog sklopa je ILI sklop.

X	Y	X ILI Y
0	0	0
0	1	1
1	0	1
1	1	1

Tablica 1: *Tablica stanja ILI operacije.*

Ovaj logički sklop je ireverzibilan, što znači da iz rezultata sklopa ne možemo jednoznačno zaključiti o ulaznim vrijednostima. Na primjer, za rezultat 1 u ILI sklopu, ne možemo sa sigurnošću reći koje su vrijednosti ulaza korištene. Ovdje dolazimo do ključne razlike u odnosu na kvantne sklopove (operatore). Kvantni sklopovi su nužno reverzibilni. Ova reverzibitnost proizlazi iz unitarnosti kvantnih sklopova, odnosno svaki sklop može biti prikazan preko unitarne matrice. Ovaj uvjet dolazi direktno od kvantne mehanike gdje s unitarnim operatorom opisujemo evoluciju kvantnih stanja.

Znači u kvantnim računalima imamo kvantne sklopove koji su reverzibilni. Ta reverzibilnost je očuvana bez obzira koliko puta primijenimo neki kvantni sklop jer su svi reverzibilni. Reverzibilnost se naruši mjerenjem jer se tada razmatrano stanje uruši u neko od svojstvenih stanja opservable koju mjerimo.

Nama poznati principi superpozicije i kvantnog ispreplitanja su od velike važnosti za kvantna računala, evo zašto. Znamo da je superpozicija linearna kombinacija dva ili više stanja koje rezultiraju u novom vektorskom stanju istog Hilbertovog prostora. Kvantna računala se grade iz kvantnih bitova za koje ćemo u tekstu koje slijedi koristiti pokratu $qubit$. Qubit, također kao klasični bit, može biti u dva stanja, $|0\rangle$ ili $|1\rangle$. Ključna razlika je da je qubit kvantno mehanički sustav, stoga se općenito kvantno stanje može prikazati kao linearna kombinacija stanja $|0\rangle$ i $|1\rangle$. Ovime qubit može biti u superpoziciji oba stanja istovremeno.

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \tag{2.1}$$

gdje su α i β amplitude vjerojatnosti. Ovo omogućava kvantnim računalima da rade s mnogo više podataka nego što omogućuju klasična računala.

Za dva kvantna stanja kažemo da su spregnuta ako nisu separabilna, odnosno mjerenje jednog sustava korelirano je s drugim sustavom. Ovaj princip se često koristi jer omogućava ispravljanje pogrešaka, olakšavanje kvantne komunikacije i povećanje učinkovitosti kvantnih algoritama [4]. U poglavlju 2.3 iznijet ćemo definiciju kvantne spregnutosti (koristi se i izraz isprepletenosti) te kasnije prikazati primjere više kvantnih spregnutih stanja.

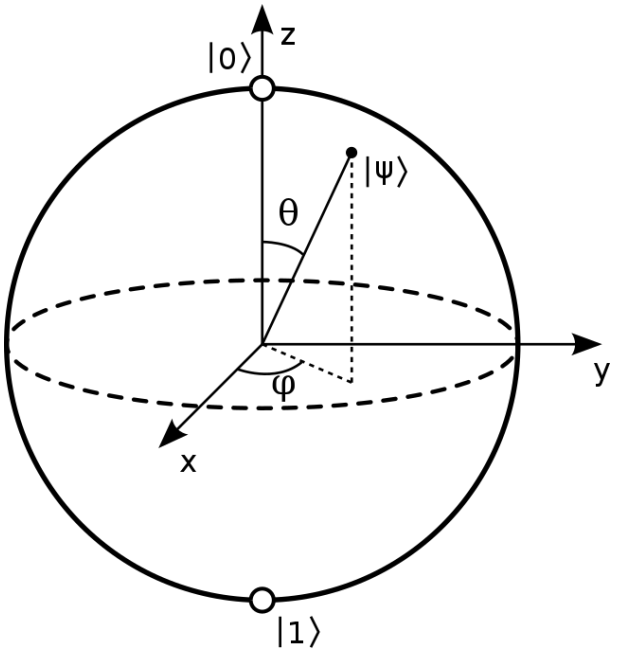
Kombinacijom svih gore navedenih principa i fizikalnom implementacijom njih dobivaju se kvantna računala. Naravno pri izgradnji se javljaju veliki izazovi poput grešaka i neželjenih šumova [5].

2.2 Prikaz stanja qubita

Stanja baze $|0\rangle$ i $|1\rangle$ reprezentiramo s matricama.

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{2.2}$$

Kako smo prije spomenuli općenito stanje jednog qubita je $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$, gdje vrijedi $|\alpha|^2 + |\beta|^2 = 1$. Definirajmo sada Blochovu sferu, koju koristimo kako bi prikazali stanje jednog qubita. Sfera reprezentira sva moguća stanja qubita, gdje svaka točka odgovara nekome stanju. Blochova sfera je jedinična sfera, gdje sjeverna polutka odgovara stanju $|0\rangle$, a južna polutka stanju $|1\rangle$. Kako bi prikazali generalno stanje qubita, koristimo vektor koji kreće od središta do neke točke na sferi, koja odgovara točno nekom stanju. Generalno stanje $|\psi\rangle$ se



Slika 1: *Prikazano generalnog stanja $|\psi\rangle$ na Blochovoj sferi [2].*

može prikazati preko θ i ϕ na sljedeći način

$$|\psi\rangle = \cos(\theta/2)|0\rangle + e^{i\phi}\sin(\theta/2)|1\rangle = \cos(\theta/2)|0\rangle + (\cos\phi + i\sin\phi)\sin(\theta/2)|1\rangle \tag{2.3}$$

gdje $0 \leq \theta \leq \pi$ i $0 \leq \phi < 2\pi$.

2.3 Stanje više qubita

Kada govorimo o stanjima više qubita podrazumijevamo tenzorski produkt između više stanja jednog qubita. Tenzorski produkt, označen simbolom $\otimes$, operacija je koja uzima dva vektora iz dva vektorska prostora i stvara novi vektor u prostoru čija je dimenzija produkt dimenzija izvornih prostora. Neka su $\mathbf{u}$ i $\mathbf{w}$ dva vektora definirana kao:

$$\mathbf{u} = \begin{pmatrix} u_1 \\ u_2 \end{pmatrix}$$
$$\mathbf{w} = \begin{pmatrix} w_1 \\ w_2 \end{pmatrix}$$

Tenzorski produkt $\mathbf{u}$ i $\mathbf{w}$ je vektor definiran kao:

$$\mathbf{u} \otimes \mathbf{w} = \begin{pmatrix} u_1 w_1 \\ u_1 w_2 \\ u_2 w_1 \\ u_2 w_2 \end{pmatrix} \tag{2.4}$$

Promotrimo ovo na primjeru stanja s dva qubita. Neka imamo dva stanja jednog qubita koja želimo kombinirati:

$$|\psi_1\rangle = a|0\rangle + b|1\rangle$$
$$|\psi_2\rangle = c|0\rangle + d|1\rangle.$$

Kombinirano stanje ova dva qubita je:

$$|\psi_1\psi_2\rangle = |\psi_1\rangle \otimes |\psi_2\rangle = ac|00\rangle + ad|01\rangle + bc|10\rangle + bd|11\rangle \tag{2.5}$$

gdje vrijedi $|ac|^2 + |ad|^2 + |bc|^2 + |bd|^2 = 1$. Konačno spomenimo još notaciju $|\psi\rangle^{\otimes k}$, koja govori da radimo tenzorski produkt stanja $|\psi\rangle$ sa samim sobom k puta. Primjerice $|\psi\rangle^{\otimes 2} = |\psi\rangle \otimes |\psi\rangle$. Vratimo se sada na kvantna spregnuta stanja i promotrimo primjer za dva qubita. Stanje dva qubita je spregnuto ako se ne može napisati kao tenzorski produkt dvaju odvojenih stanja qubita. Drugim riječima, ako ne možemo napisati stanje u obliku $|\psi_1\rangle \otimes |\psi_2\rangle$, onda je to

stanje spregnuto. Promotrimo sljedeće stanje koje je spregnuto stanje dva qubita

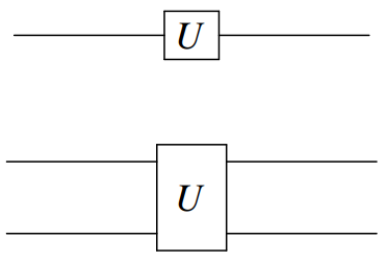
$$\left|\Phi^{+}\right\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle). \tag{2.6}$$

Stanje iz (2.6) nazivamo prvim Bellovim stanjem, više ovakvih stanja biti će prikazano kasnije. Kako bi se bolje shvatila kvantna spregnutost možemo promotriti što se događa kada mjerimo oba qubita u stanju (2.6), a zatim u stanju (2.5). Kod stanja (2.6) ako izmjerimo prvi qubit i dobijemo rezultat $|0\rangle$, odmah znamo da će mjerenje drugog qubita također dati $|0\rangle$, iako nismo izravno mjerili drugi qubit. Slično, ako izmjerimo prvi qubit i dobijemo $|1\rangle$, drugi qubit će sigurno biti u stanju $|1\rangle$ ako ga izmjerimo.
Dok kod ne spregnutog stanja (2.5), ako mjerenjem prvog qubita dobijemo stanje $|0\rangle$ onda ćemo mjerenjem drugi qubit naći u stanju $|0\rangle$ s vjerojatnosti $|c|^2$ ili u stanju $|1\rangle$ s vjerojatnosti $|d|^2$. Odnosno mjerenjem jednog qubita ne znamo odmah stanje drugog qubita. Iz ovog jednostavnog primjera se može uočiti važnost kvantno spregnutih stanja.

Kvantnim sklopovima (nazivamo ih i kvantnim operatorima) djelujemo na qubite i tako mijenjamo njihovo stanje. Prije nego što navedemo razne vrste operatora prođimo kroz kvantne sklopovske dijagrame koje koristimo kako bi grafički prikazali kvantni algoritam. Kvantni algoritam dobijemo djelovanjem jednog ili više kvantnih operatora. Kvantne sklopovske dijagrame crtamo na sljedeći način, počinjemo uvijek s linijom, ako je linija prazna, nema operatora na njoj, stanje ostaje nepromijenjeno (slika 2). Inicijalno pripremljeno stanje se označava s ketom na lijevoj strani linije. Unarne operatore, one koji djeluju na jedan qubit, označavamo s

Slika 2: *Ostavljamo stanje $|0\rangle$ nepromijenjeno [2].*

kvadratom unutar kojeg je slovo koje reprezentira taj operator. Sklopove koje djeluju na dva qubita, odnosno binarne sklopove, s kvadratom koji se proteže preko dvije linije (slika 3).



Slika 3: *Sklop koji djeluje na jedan qubit (gore), sklop koji djeluje na dva qubita (dolje) [2].*

2.4 Unarni operatori

Tri unarna operatora se mogu dobiti direktno iz Paulijevih matrica. Prvi je NE operator, koji djeluje kao i NE operator u klasičnim sklopovima. Označavamo ga slovom X.

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \tag{2.7}$$

Ako operatorom X djelujemo na stanje $|0\rangle$. Dobijemo

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

Dalje imamo Y operator, te operator Z

$$Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{2.8}$$

Ova dva operatora rotiraju vektor, redom oko Y osi te oko Z osi. Ove dvije osi odnose se na osi na Blochovoj sferi, koja je zgodan način vizualiziranja stanja qubita.
Uvedimo generalni operator s kojim mijenjao fazu. Kada ovim operatorom djelujemo na stanje $|0\rangle$, ono ostaje nepromijenjeno, a stanje $|1\rangle$ se rotira za kut ϕ.

$$R_\varphi := \begin{pmatrix} 1 & 0 \\ 0 & e^{i\varphi} \end{pmatrix} \tag{2.9}$$

Primijetimo da je operator Z specijalni slučaj operatora R_ϕ kada je $\phi = \pi$. Uvode se još dva operatora koji su specijalni slučaj od R_ϕ operatora, to su S operator, gdje je $\phi = \pi/2$, te T operator gdje je $\phi = \pi/4$.
Sljedeći operator koji ćemo proći je od velike važnosti za kvantno računarstvo, omogućava da qubit u jednom od stanja baze (2.2) prikažemo kao linearnu kombinaciju stanja baze. Zove se Hadamardov operator.

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{2.10}$$

Pogledajmo kako djeluje na stanja $|0\rangle$ i $|1\rangle$.

$$\frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1+0 \\ 1+0 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \frac{|0\rangle + |1\rangle}{\sqrt{2}} \tag{2.11}$$

$$\frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 0+1 \\ 0-1 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -1 \end{pmatrix} = \frac{|0\rangle - |1\rangle}{\sqrt{2}} \tag{2.12}$$

Vidimo da H operator stanje baze projicira u superpoziciju stanja baze. Još treba navesti da postoji operator identiteta koji ostavlja stanje nepromijenjeno.

2.5 Binarni operatori

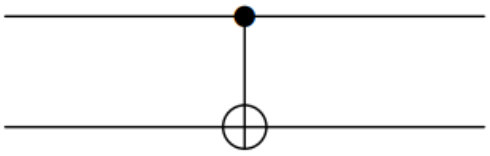
U sustavu dva qubita koristimo, po konvenciji, sljedeća stanja baze

$$|00\rangle = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad |01\rangle = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \quad |10\rangle = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}, \quad |11\rangle = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \tag{2.13}$$

Vrlo bitan binarni operator je CNOT operator. U ovome operatoru prvi qubit identificiramo kao kontrolni qubit, a drugi kao ciljani qubit. Ako je kontrolni qubit u stanju $|0\rangle$ onda ne činimo ništa s ciljanim qubitom, no ako je kontrolni qubit u stanju $|1\rangle$ onda primjenjujemo NE(NOT) operator na drugi qubit. Ovaj operator se koristi kako bi se dva qubita kvantno spregnula. Prikazan je na slici 4.

$$CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{2.14}$$

CNOT operatori je i bitan za koncept računalne univerzalnosti u kvantom računanju. Klasično,



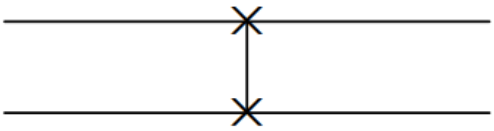
Slika 4: *Prikaz CNOT operatora [2].*

korištenjem NAND sklopa možemo izgraditi svo ostalo klasično sklopovlje [2]. Kod kvantnih računala univerzalnost se postiže sa skupom operatora CNOT, S, H. Univerzalnost zapravo predstavlja skup sklopovlja pomoću kojeg možemo doći do bilo koje točke na Blochovoj sferi.

Sljedeći binarni operator kojeg prikazujemo je SWAP operator, on jednostavno zamijeni stanje $|01\rangle$ u $|10\rangle$ i obrnuto. Prikazan je na slici 5.

$$SWAP = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.15}$$

Spomenimo još CZ operator, gdje kao kod CNOT-a imamo kontrolni i ciljni qubit. Ako je

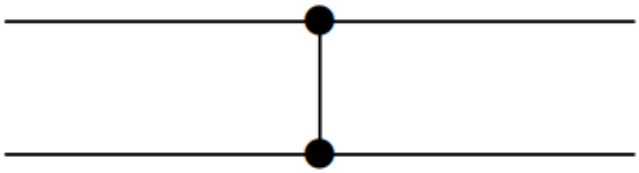


Slika 5: *Prikaz SWAP operatora [2].*

kontrolni qubit u stanju $|1\rangle$ onda na ciljani primjenjujemo Z operator. Prikazana je na slici 6 Matrica operatora je sljedeća.

$$CZ = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix} \tag{2.16}$$

Još se može uočiti da je CZ sklop simetričan, rezultat ne ovisi o tome koji qubit izaberemo kao



Slika 6: *Prikaz CZ operatora [2].*

kontrolni.

2.6 Bellova stanja

Bellova stanja su specifična kvantna stanja dva qubita koja predstavljaju najjednostavnije (i maksimalne) primjere kvantne isprepletenosti.

$$\begin{aligned} \left|\Phi^+\right\rangle &= \frac{|00\rangle + |11\rangle}{\sqrt{2}} \\ \left|\Phi^-\right\rangle &= \frac{|00\rangle - |11\rangle}{\sqrt{2}} \\ \left|\Psi^+\right\rangle &= \frac{|01\rangle + |10\rangle}{\sqrt{2}} \\ \left|\Psi^-\right\rangle &= \frac{|01\rangle - |10\rangle}{\sqrt{2}} \end{aligned} \tag{2.17}$$

Prođimo detaljno kroz konstrukciju prvog Bellovog stanja $|\Phi^+\rangle = \frac{|00\rangle+|11\rangle}{\sqrt{2}}$.

1. **Bellovo stanje** $|\Phi^+\rangle$: Početno stanje je $|00\rangle = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$. Zatim djelujemo Hadamardovim operatorom na prvi qubit. Kako bi razumjeli djelovanje Hadamardovog operatora na qubit koji je dio stanja s dva qubita, pogledajmo tenzorski produkt $\mathrm{H} \otimes I$. Ovime dobijemo proširenu matricu gdje na prvi qubit djelujemo s Hadamardovim operatorom, a na drugi matricom identiteta.

$$\mathrm{H} \otimes I = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{pmatrix}$$

Dobivenom matricom djelujemo na stanje $|00\rangle$.

$$\frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

Zatim primjenjujemo CNOT sklop i konačno dolazimo do prvog Bellovog stanja.

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \frac{|00\rangle + |11\rangle}{\sqrt{2}}$$

2. **Bellovo stanje** $|\Phi^-\rangle$: Drugo Bellovo stanje jednako je prvom samo ima negativnu fazu. Neka ponovno krećemo od stanja $|00\rangle$. Prvo djelujemo operatorom X na prvi qubit čime dobijemo stanje $|10\rangle$. Zatim je proces jednak kao za prvo Bellovo stanje. Djelujemo Hadamardovim operatorom na prvi qubit pa dobijemo

$$\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \otimes |0\rangle = \frac{1}{\sqrt{2}}(|00\rangle - |10\rangle).$$

Konačno, djelujemo CNOT operatorom gdje je prvi qubit kontrolni, pa dobijemo

$$\frac{1}{\sqrt{2}}(|00\rangle - |11\rangle).$$

3. **Bellovo stanje** $|\Psi^+\rangle$: Krećemo ponovno od stanja $|00\rangle$. Primijenimo X operator na drugi

qubit pa imamo $|01\rangle$. Zatim djelujemo Hadamardovim operatorom na prvi qubit pa imamo

$$\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes |1\rangle = \frac{1}{\sqrt{2}}(|01\rangle + |11\rangle).$$

Konačno djelujemo CNOT operatorom gdje je prvi qubit kontrolni pa dobijemo

$$\frac{1}{\sqrt{2}}(|01\rangle + |10\rangle).$$

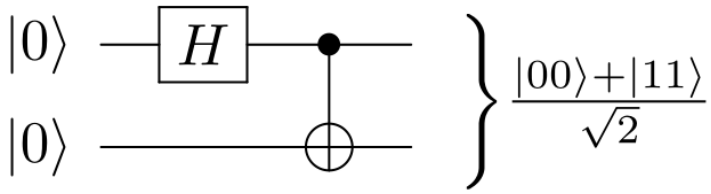
4. **Bellovo stanje** $|\Psi^-\rangle$: Isto kao treće stanje samo ima negativnu fazu. Na prvi i drugi qubit stanja $|00\rangle$ djelujemo X operatorom, čime dobijemo $|11\rangle$. Zatim djelujemo Hadamardovim operatorom na prvi qubit, imamo

$$\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \otimes |1\rangle = \frac{1}{\sqrt{2}}(|01\rangle - |11\rangle).$$

Konačno još CNOT operator gdje je prvi qubit kontrolni, pa imamo

$$\frac{1}{\sqrt{2}}(|01\rangle - |10\rangle).$$

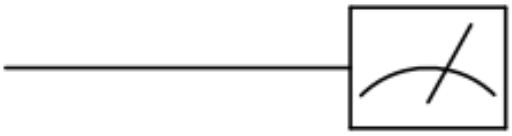
Možemo još pogledati vizualnu reprezentaciju za stanje $|\Phi^+\rangle$ na slici 7.



Slika 7: *Vizualna reprezentacija sklopovlja koje daje $|\Phi^+\rangle$ [3].*

2.7 Mjerenja na kvantnim računalima

U kvantnom računarstvu, mjerenje se često obavlja na kraju kvantnog algoritma kako bi se dobili konačni rezultati. Prije mjerenja, kvantni algoritmi koriste unitarne operatore za manipulaciju kvantnim stanjima i izvođenje računskih operacija. Kako je prije navedeno, unitarni operatori osiguravaju da se kvantno stanje očuva tijekom izvođenja algoritma, a mjerenje se koristi za dobivanje informacija iz tog stanja. Mjerenje je ireverzibilan proces. Prema Bornovom pravilu, vjerojatnost da se kvantni sustav nalazi u određenom stanju, koje se može izmjeriti, proporcionalna je kvadratu apsolutne vrijednosti amplitude vjerojatnosti tog stanja. Stoga Bornovo pravilo omogućuje opisivanje vjerojatnosti rezultata mjerenja, dok unitarni operatori omogućuju izvođenje računskih operacija nad kvantnim stanjima. Oznaka da se mjeri u kvantom krugu prikazan je na slici 8.



Slika 8: *Oznaka mjerenja [2].*

3 Izgradnja sklopova i jednostavnih algoritama pomoću Qiskita-a

Qiskit je otvoreni okvir za kvantno računanje razvijen od strane IBM-a. Pruža moćnu platformu za dizajniranje, simuliranje i izvođenje kvantnih algoritama na kvantnim računalima i simulatorima [6].

3.1 Hadamardov operator

Izgradimo pomoću Qiskita vrlo jednostavnu simulaciju gdje na qubit u stanju $|0\rangle$ primijenimo Haadmardov operator te potom napravimo mjerenje. Mjerenje izvršimo 100 000 puta.
Ovdje uvozimo već napravljene module Qiskita koji nam omogućavaju izgradnju kvantnog kruga. Koristimo klasu `AerSimulator` preko kojeg simuliramo rad kvantnog računala na klasičnom računalu. Još uvozimo klase `QuantumCircuit` preko koje gradimo kvantni krug i funkciju `transpile` preko koje se izvodi transpilacija, odnosno kvantni krug se prilagođava pojedinoj pozadini izvođenja. Primjerice pozadina izvođenja može biti pravo kvantno računalo pa tada krug treba prilagoditi tom specifičnom hardveru. Konkretno, ovdje to prilagođavamo simulatoru [6].

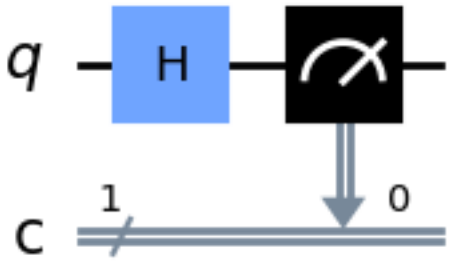
```
from qiskit import QuantumCircuit, transpile
from qiskit.visualization import plot_histogram #vizualizacija rezultata
from qiskit.providers.aer import AerSimulator
```

Ovdje definiramo kvantni krug koji ima jedan qubit i jedan klasični bit. Na qubitu primjenjujemo operatore dok klasični bit koristimo za spremanje rezultata mjerenja. Potom definiramo strukturu kvantnog kruga, djelujemo Hadamardovim operatorom i zatim postavljamo operaciju mjerenja.

```
# Definiranje kruga
circuit = QuantumCircuit(1, 1)  #1 qubit i 1 klasicni bit za spremanje
                                    rezultata mjerenja
circuit.h(0)  #Primjena Hadamardovog operatora na qubit na pozicji 0
circuit.measure(0, 0)  #Izvrsavanje mjerenja na qubitu na poziciji 0 i
                                    spremanje podataka u klasicni bit na
```

```
                                        pozciji 0.
```

Sada crtamo dobiveni kvantni krug. Rezultat je prikazan na slici 9.



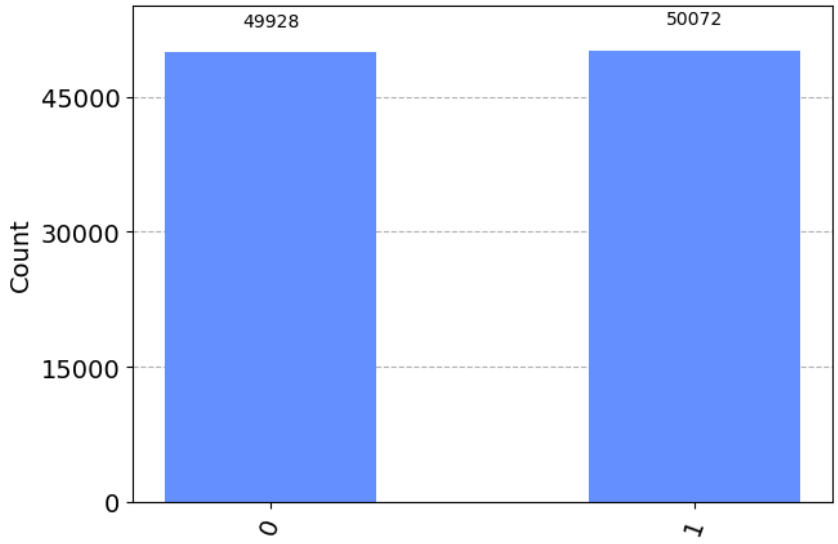
Slika 9: *Prikaz kvantnog kruga koji se sastoji od Hadamardovog operatora, mjerenja i klasičnog bita za spremanje rezultata mjerenja.*

```
#Crtanje kvantnog kruga
display(circuit.draw('mpl'))
```

Sada je potrebno simulirati napravljeni kvantni krug. Definiramo objekt `simulator` klase `AerSimulator`. Izvršavamo još transpilaciju, pa potom pomoću funkcije `simulator.run(compiled_circuit, shots=100000).result()` dobivamo rezultate izvršavanja simulacije. Argument `shots` postavljamo na 100000, čime govorimo simulaciji da napravi 100000 mjerenja [6]. Konačno još ispisujemo rezultate mjerenja i crtamo histogram dobivenih rezultata.

```
# Simulacija kruga
simulator = AerSimulator()
compiled_circuit = transpile(circuit, simulator) #transpilacija
rezultat = simulator.run(compiled_circuit, shots=100000).result()

# Ispis rezultata mjerenja
print("\nRezultati mjerenja:")
print(rezultat.get_counts())

#Crtanje histograma
plot_histogram(rezultat.get_counts())
```

Promotrimo dobivene rezultate.
Slika 10 prikazuje histogram s rezultatima mjerenja. Djelovanjem Hadamardovim operatorom na stanje $|0\rangle$ dobijemo stanje $\frac{|0\rangle}{\sqrt{2}} + \frac{|1\rangle}{\sqrt{2}}$. Iz čega vidimo da za amplitude vjerojatnosti vrijedi $|\alpha|^2 = |\beta|^2 = \frac{1}{2}$. Ovo nam govori da imamo 50% vjerojatnosti da nakon mjerenja sustav bude



Slika 10: *Histogram rezultata 100 000 mjerenja na kvantom krugu koji se sastoji od Hadamardovog operatora.*

u stanju $|0\rangle$ tako i za stanje $|1\rangle$. Ovo možemo i potvrditi iz histograma gdje vidimo da smo 49.928% puta dobili stanje $|0\rangle$, a 50.072% stanje $|1\rangle$ za 100000 mjerenja.

3.2 Izgradnja Bellovg stanja Φ^+

Konstruirajmo sada Bellovo stanje $|\Phi^+\rangle = \frac{|00\rangle+|11\rangle}{\sqrt{2}}$ pomoću Qiskita. Uvodimo iste module kao u prošlom primjeru. Kako je prije bilo pokazano Bellova stanja su stanja dva qubita pa tako koristimo dva qubita pri inicijalizaciji kvantnog kruga. Koristimo jedan klasični bit za spremanje rezultata mjerenja.

```
#uvozimo iste module kao u proslom primjeru
from qiskit import QuantumCircuit, transpile
from qiskit.providers.aer import AerSimulator
from qiskit.visualization import plot_histogram

#Definiranje kvantnog kruga
circuit = QuantumCircuit(2, 1)
```

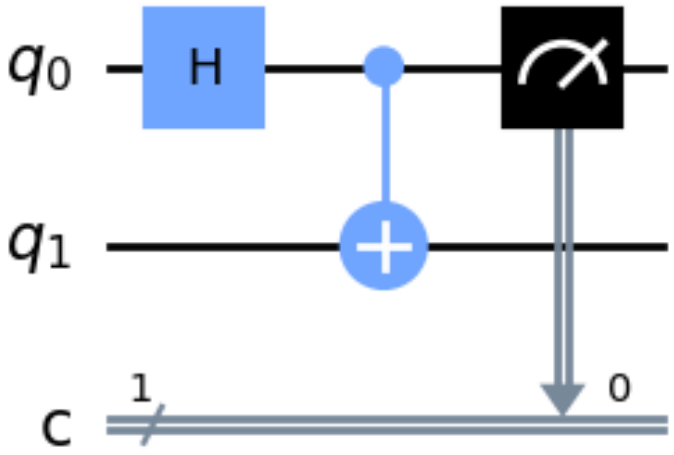
Potom djelujemo Hadamardovim operatorom na prvi qubit s naredbom `circuit.h(0)`, zatim CNOT (u Qiskitu oznaka CX) operatorom na oba qubita s naredbom `circuit.cx(0, 1)` gdje je prvi argument kontrolni qubit, a drugi ciljani. Konačno, dodajemo operaciju mjerenja u kvantni krug. Kako se radi o spregnutom Bellovom stanju dovoljno je mjeriti samo jedan qubit odmah znamo onda i stanje drugog.

```
#Dodaj operatore za pripremu Bellovog stanja
circuit.h(0)
circuit.cx(0, 1) #prvi agrument je kontrolni qubit, drugi argument je
                                        ciljni qubit
#Mjerenje
circuit.measure(0, 0)
```

Na slici 11 prikazan je dobiveni krug. Potom izvršavamo simulaciju na isti način kao u prošlom primjeru. Konačno još ispisujemo rezultate i radimo histogram dobivenih mjerenja.

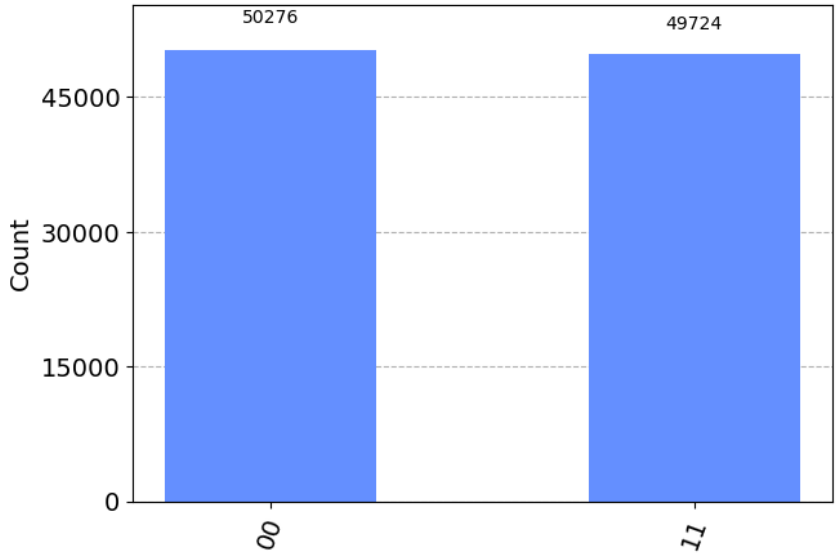
```
#Nacrtaj krug
display(circuit.draw('mpl'))
#Simuliraj krug koristeci AerSimulator
simulator = AerSimulator()
compiled_circuit = transpile(circuit, simulator)
res = simulator.run(compiled_circuit, shots=100000).result()

#Ispisi rezultate
print("\nMjerenja:")
print(res.get_counts())
plot_histogram(res.get_counts())
```

Kako su Bellova stanja spregnuta stanja prvog i drugog qubita, ako izmjerimo prvi qubit stanja $|\Phi^+\rangle$ i pronađemo ga u stanju $|0\rangle$, drugi qubit će se također odmah srušiti u stanje $|0\rangle$. Slično



Slika 11: *Kvantni krug koji koristimo za izgradnju Bellovog stanja* Φ^+.

tome, ako pronađemo prvi qubit u stanju $|1\rangle$, drugi qubit će se srušiti u stanje $|1\rangle$. S obzirom na ovo u kodu poviše smo i mogli mjeriti samo jedan qubiti i biti sigurni za rezultat drugog qubita. Pogledajmo dobivene rezultate. Iz slike 12 vidimo da su mjerenja ponovno u skladu s amplitu-



Slika 12: *Histogram s rezultatima 100 000 mjerenja Bellovog stanja* Φ^+.

dama vjerojatnosti za stanja $|00\rangle$ i $|11\rangle$.

3.3 Implementacija kvantnog teleportiranja u Qiskitu

Kvantna teleportacija je protokol koji iznimne važnosti za kvantna računala. Ovaj protokol omogućava prijenos stanja između udaljenih qubita bez da se taj sam qubit prenese. Koristi principe kvantne spregnutosti i superpozicije [4]. Prođimo sada kroz korake ovog protokola. Koristit će se klasična notacija za informacijske protokole, gdje je pošiljatelj Alice, a primatelja Bob.
Sustav ima tri qubita:

- q_A: Qubit koji Alice želi poslati $|\psi\rangle$.
- $q_{A'}$: Alicein dio spregnutog Bellovog stanja.
- q_B: Bobov dio spregnutog Bellovog stanja.

Alice i Bob dijele spregnuto Bellovo stanje, u tom stanju su spregnuti $q_{A'}$ i q_B qubit.

$$\left|\Phi^+\right\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) \tag{3.1}$$

Nadalje Alice želi poslati qubit q_A koji je u sljedećem stanju

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \tag{3.2}$$

gdje naravno vrijedi $|\alpha|^2 + |\beta|^2 = 1$.
Početno stanje je dakle

$$|\psi\rangle\otimes\left|\Phi^+\right\rangle = \frac{1}{\sqrt{2}}(\alpha|0\rangle(|00\rangle+|11\rangle)+\beta|1\rangle(|00\rangle+|11\rangle)) = \frac{1}{\sqrt{2}}(\alpha|000\rangle+\alpha|011\rangle+\beta|100\rangle+\beta|111\rangle).$$

Ovo je ukupno stanje svih triju qubita.
Zatim Alice primjenjuje CNOT operator s q_A kao kontrolnim i $q_{A'}$ kao ciljanim qubitom, pa imamo

$$\frac{1}{\sqrt{2}}(\alpha|000\rangle + \alpha|011\rangle + \beta|110\rangle + \beta|101\rangle). \tag{3.3}$$

Zatim Hadamardovim operatorom djelujemo na q_A, ukupno stanje sustava postaje:

$$\begin{aligned}\frac{1}{\sqrt{2}}\Bigg[\alpha\left(\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)|00\rangle\right) + \alpha\left(\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)|11\rangle\right)\\ +\beta\left(\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)|10\rangle\right) + \beta\left(\frac{1}{\sqrt{2}}|0\rangle - |1\rangle\right)|01\rangle\Bigg]\end{aligned} \tag{3.4}$$

Sređivanjem dolazimo do:

$$\frac{1}{2}\left(\alpha|000\rangle + \alpha|011\rangle + \alpha|100\rangle + \alpha|111\rangle + \beta|001\rangle + \beta|010\rangle - \beta|101\rangle - \beta|110\rangle\right) \tag{3.5}$$

Informacija o početnom stanju $|\psi\rangle$ sada se "nalazi" u cijelom sustavu, a stanje q_B (Bobov qubit) ovisi o rezultatima mjerenja qubita q_A i $q_{A'}$.
Potom Alice izvršava mjerenje oba qubita q_A i $q_{A'}$ gdje rezultate mjerenja spremi u dva klasična bita m_1 i m_2, gdje m_1 odgovara mjerenju na qubitu q_A, a m_2 na qubitu $q_{A'}$. Alice potom klasičnim putem Bobu šalje ta dva bita. Primijetimo da ima četiri moguća ishoda mjerenja, a to su: 00, 01, 10 i 11 ($m_1 m_2$).
Razmotrimo svaku od četiri mogućnosti i vidimo kako se stanje q_B mijenja u svakom slučaju.

1. **Ako Alice izmjeri 00:**

   Nakon mjerenja 00, sustav se uruši u:

   $$\alpha|000\rangle + \beta|001\rangle$$

   Ovdje, q_B je u stanju $\alpha|0\rangle + \beta|1\rangle$. Dakle, Bob ne treba primijeniti nikakve operacije.

2. **Ako Alice izmjeri 01:**

   Nakon mjerenja 01, sustav se uruši u:

   $$\alpha|011\rangle + \beta|010\rangle$$

   Ovdje, q_B je u stanju $\alpha|1\rangle + \beta|0\rangle$. Da bi se vratio u početno stanje $\alpha|0\rangle + \beta|1\rangle$, Bob treba primijeniti X operator.

3. **Ako Alice izmjeri 10:**

   Nakon mjerenja 10, sustav se uruši u:

   $$\alpha|100\rangle - \beta|101\rangle$$

   Ovdje, q_B je u stanju $\alpha|0\rangle - \beta|1\rangle$. Da bi se vratio u početno stanje $\alpha|0\rangle + \beta|1\rangle$, Bob treba primijeniti Z operator.

4. **Ako Alice izmjeri 11:**

   Nakon mjerenja 11, sustav se uruši u:

   $$\alpha|111\rangle - \beta|110\rangle$$

   Ovdje, q_B je u stanju $\alpha|1\rangle - \beta|0\rangle$. Da bi se vratio u početno stanje $\alpha|0\rangle + \beta|1\rangle$, Bob prvo treba primijeniti X operator, a zatim Z operator.

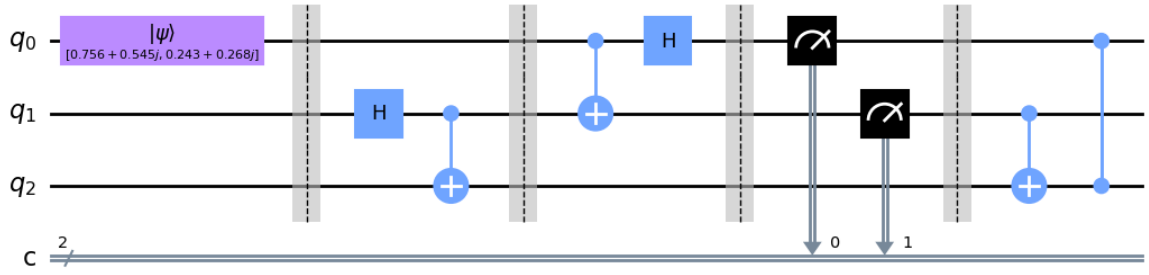
Nakon ovoga se Bobov qubit transformira u stanje $|\psi\rangle$, čime završava teleportacija. Napomenimo još kako ovo nije kopiranje kvantnog stanja $|\psi\rangle$ s obzirom da Alice mjerenjem to stanje uništi i

time nije narušen teorem o nemogućnosti kloriranja kvantnog stanja [2].
Sada kada smo prošli kroz algoritam pogledajmo njegovu implementaciju u Qiskitu, čiji je kvantni krug prikazan na slici 13.
Ponovno započinjemo s uvozom modula. Ovdje umjesto `AerSimulator` klase koristimo `Aer` klasu kako tu možemo odabrati da nam simulator bude `statevector_simulator` [6]. Ova vrsta simulacije egzaktno računa sva stanja i naravno nije moguća na kvantom računalu. Ovdje to koristimo kako bi pokazali da se finalno stanje uspješno pretvori u početno.

```
from qiskit import QuantumCircuit, Aer, transpile
from qiskit.visualization import plot_bloch_multivector
from qiskit.extensions import Initialize #ovo koristimo kada zelimo
                                         postaviti qubit u odredeno stanje
                                         koje nije stanje baze
import numpy as np
```

Zatim definiramo funkciju za kvantu teleportaciju. Tu stvaramo kvantni krug koji se sastoji od tri qubita, prvi qubit je q_A, drugi $q_{A'}$ i treći q_B. Koristimo dva klasična qubita za mjerenje. Potom dodajemo operatore kako je prethodno objašnjeno.

```
def quantum_teleportation(psi): #kao argument prima kvanto stanje koje
                                zelimo teleportirati
  init_gate = Initialize(psi)

  # Kreiranje kvantnog sklopa
  qc = QuantumCircuit(3, 2)

  # Inicijalizacija Aliceinog qubita
  qc.append(init_gate, [0])
  qc.barrier() #ovo koristimo kako bi vizualno kvantni krug bio pregledniji
  #ovdje radimo Bellovo stanje izmedu qubita 1 i 2, 1 = q_A', 2 = q_B
  qc.h(1)
  qc.cx(1, 2)
  qc.barrier()
  #ovdje izvodimo proces koji je detaljno objasnjen u teorijskom dijelu
  qc.cx(0, 1)
  qc.h(0)
  qc.barrier()
  qc.measure([0, 1], [0, 1]) #izvodimo mjerenja
  qc.barrier()
  #dodajemo operatore koje Bob mora iskoristiti kako bi dosao do pocetnog
                                stanja
  qc.cx(1, 2)
  qc.cz(0, 2)

  #Dobivanje konacnog kvantog stanja
  aer_sim_statevector = Aer.get_backend('statevector_simulator') #
                                dohvacanje simulatora
```



Slika 13: *Kvantni krug protokola kvante teleportacije.*

```
  t_qc_statevector = transpile(qc, aer_sim_statevector)
  result_statevector = aer_sim_statevector.run(t_qc_statevector).result() #
                                        izvrsavanje simulacije
  final_state = result_statevector.get_statevector() #spremanje konacnog
                                        vektora

  display(qc.draw("mpl")) #crtanje kvantnog kruga

  return final_state
```

Još definiramo funkciju za generiranje slučajnog početnog stanja.

```
def generate_random_state():
  import numpy as np
  #Generiraj nasumicne realne i imaginarne dijelove za dvije amplitude
  real_components = np.random.rand(2) * 2 - 1
  imaginary_components = np.random.rand(2) * 2 - 1

  #Kombiniraj realne i imaginarni dijelove da formiras kompleksne amplitude
  complex_amplitudes = real_components + 1j*imaginary_components

  #Normalizacija
  norm_factor = np.linalg.norm(complex_amplitudes)
  normalized_amplitudes = complex_amplitudes / norm_factor

  return normalized_amplitudes
```

Konačno izvršavamo prethodno definirane funkcije i ispisujemo rezultate i vizualizaciju preko Blochove sfere (slika 14 i 15).

```
# Definiraj i normaliziraj pocetno stanje
psi = generate_random_state()


# Izvrsi kvantnu teleportaciju
final_state = quantum_teleportation(psi)
```

```
# Ispisi i vizualiziraj rezultate
print("Pocetno stanje koje Alice zeli teleportirati:", psi)
bob_state_0 = sum(final_state[i] for i in [0, 1, 2, 3])
bob_state_1 = sum(final_state[i] for i in [4, 5, 6, 7])
bob_state = [bob_state_0, bob_state_1]

print("Konacno stanje koje Bob prima:", bob_state)

# Vizualizacija
display(plot_bloch_multivector(psi))
display(plot_bloch_multivector(final_state))
```

Za rezultat dobijemo:

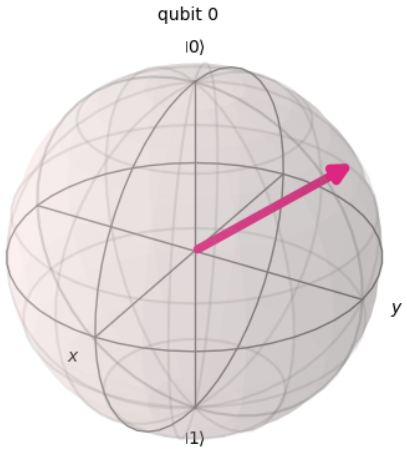
Početno stanje koje Alice želi teleportirati:

$(-0.66571578 - 0.5678248i,\ 0.45843943 - 0.15566243i)$

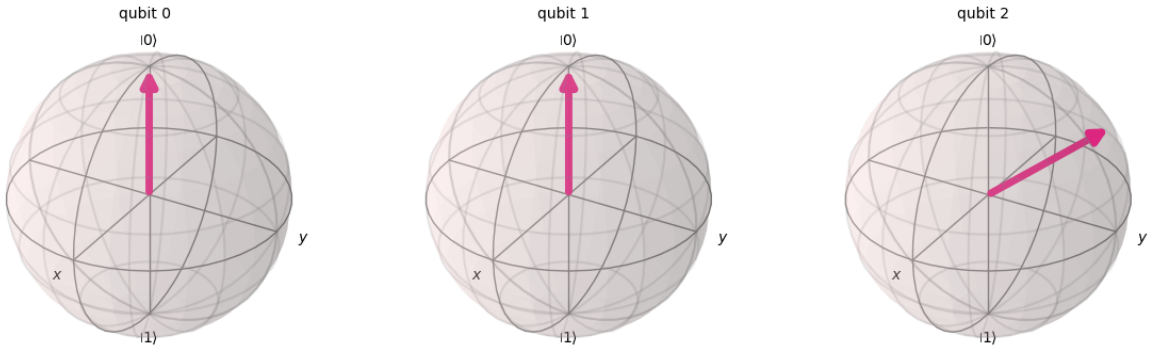
Konačno stanje koje Bob prima:

$(-0.66571578 - 0.5678248i,\ 0.45843943 - 0.15566243i)$

Vidimo da dobijemo potpuno slaganje. Promotrimo još rezultate grafički. Na slici 14 prikazana je Blochova sfera početnog stanja koje Alice želi teleportirati, odnosno stanje qubita q_A. Usporedbom sa slikom 15 možemo uočiti jednakost između Alicinog početnog stanja i konačnog Bobovog stanja q_B. Primijetimo i kako je u konačnom stanju na slici 15 Alicin qubit q_A više nije u početnom stanju već je u stanju baze.



Slika 14: *Početno stanje Alicinog qubita koje želimo teleportirati. Ovo stanje je nasumično dobiveno, konstruiranim generatorom.*



Slika 15: *Konačno stanje sva tri qubita. Primijetimo da je treći (Bobov) qubit jednak stanju sa slike 14. Također primijetimo kako je ovdje stanje prvog qubita različito od njegovog početnog stanja što je u skladu s teoremom o nemogućnosti klonirana kvantnih stanja.*

4 VQE algoritam

Kako je već spomenuto u uvodu kvantni VQE algoritam trenutačno je jedan od najperspektivnijih algoritama za rad na današnjim kvantnim računalima. Popularni algoritmi poput Schorovog algoritma za pronalaženje prostih faktora cijelog broj jednostavno još nisu mogući na današnjim uređajima [3]. U ovom dijelu proći ćemo kroz teorijsku osnovu za VQE algoritam. U nekim dijelovima biti će navedene samo neke od danas korištenih metoda s obzirom da ih ima mnogo i da se pojedini dijelovi ovog algoritma konstantno dalje unaprjeđuju.

4.1 Definicija VQE-a

VQE je baziran na varijacijskoj metodi kvantne mehanike koja se koristi kada je hamiltonijan poznat, ali pripadajuću Schrodingerovu jednadžbu ne možemo riješiti ni egzaktno ni korištenjem računa smetnje. Varijacijska metoda je korisna za dobivanje gornje granice svojstvenih energija. Kod varijacijske metode ne rješavamo direktno svojstveni problem

$$\hat{H}|\psi\rangle = E|\psi\rangle \tag{4.1}$$

već se traže aproksimativne vrijednosti svojstvenih stanja i energija iz varijacijske jednadžbe

$$\delta E(\psi) = 0 \tag{4.2}$$

gdje je $E(\psi)$ očekivana vrijednost za stanje $|\psi\rangle$,

$$E(\psi) = \frac{\langle\psi|\hat{H}|\psi\rangle}{\langle\psi \mid \psi\rangle}. \tag{4.3}$$

Ako postavimo da $|\psi\rangle$ ovisi o nekom parametru θ tada će i $E(\psi)$ ovisiti o njemu. Parametar θ možemo prilagođavati tako da minimiziramo $E(\psi)$. Bitno je još naglasiti da za bilo koju probnu funkciju $|\psi\rangle$ uvijek vrijedi

$$E_0 \leqslant \frac{\langle\psi|\hat{H}|\psi\rangle}{\langle\psi \mid \psi\rangle}. \tag{4.4}$$

S obzirom da je VQE baziran na varijacijskoj metodi, zadatak VQE je također pronaći parametrizaciju od $|\psi\rangle$ tako da očekivana vrijednost hamiltonijana bude minimalna. Kako se ovo izvršava na kvantnim računalima bitno je problem iskazati tako da se može izvršiti na njima. S obzirom da na kvantnim računalima radimo s unitarnim operatorima, stanje $|\psi\rangle$ prikazujemo kao djelovanje općenito parametriziranog unitarnog operatora $U(\boldsymbol{\theta})$ na početno stanje N qubita, gdje $\boldsymbol{\theta}$ označava skup parametara koji imaju vrijednosti od $(-\pi, \pi]$. Početni qubiti su svi inicijalizirani na $|0\rangle$ pa ih sve prikazujemo preko $|\mathbf{0}\rangle$. Napomenimo još da $|\psi\rangle$ obavezno normalizirana tako da problem minimizacije pišemo na sljedeći način

$$E_{\mathrm{VQE}} = \min_{\boldsymbol{\theta}} \left\langle \mathbf{0} \left| U^\dagger(\boldsymbol{\theta}) \hat{H} U(\boldsymbol{\theta}) \right| \mathbf{0} \right\rangle. \tag{4.5}$$

Jednadžba (4.5) se često naziva funkcija troška, naziv koji potječe iz strojnog učenja. Još preostaje zapisati hamiltonijan kako bi bio moguć izračun na kvantnim računalima. Hamiltonijan pišemo na način da je direktno mjerljiv na kvantnim računalima. Ovo postižemo preko tenzorskog produkta Paulijevih operatora. Kasnije će točno biti prikazan način kako ovo postići. U potpoglavlju 2.3 uveli smo oznaku $|\psi\rangle^{\otimes k}$, nju sada proširujemo na

$$\hat{P}_a \in \{I, X, Y, Z\}^{\otimes N} \tag{4.6}$$

gdje su I, X, Y i Z Paulijevi operatori, a N predstavlja broj qubita korišten pri odabiru probne funkcije. Ovim dobivamo tenzorski produkt koji se sastoji o N Paulijevih operatora. Primjerice za $N = 3$ neki $\hat{P}_a$ bi mogao biti $I \otimes Y \otimes Z$, odnosno moguća je bilo koja kombinacija Paulijevih operatora. Općeniti hamiltonijan stoga pišemo

$$\hat{H} = \sum_a^{\mathcal{P}} w_a \hat{P}_a \tag{4.7}$$

gdje w_a predstavlja težine, a $\mathcal{P}$ broj tenzorskih produkta. Jednadžba (4.5) postaje

$$E_{\mathrm{VQE}} = \min_{\boldsymbol{\theta}} \sum_a^{\mathcal{P}} w_a \left\langle \mathbf{0} \left| U^\dagger(\boldsymbol{\theta}) \hat{P}_a U(\boldsymbol{\theta}) \right| \mathbf{0} \right\rangle. \tag{4.8}$$

Iz ovog izraza se može točno uočiti dvojnost VQE algoritma. Očekivana vrijednost za svaki pojedini tenzorski produkt

$$E_{P_a} = \left\langle \mathbf{0} \left| U^\dagger(\boldsymbol{\theta}) \hat{P}_a U(\boldsymbol{\theta}) \right| \mathbf{0} \right\rangle \tag{4.9}$$

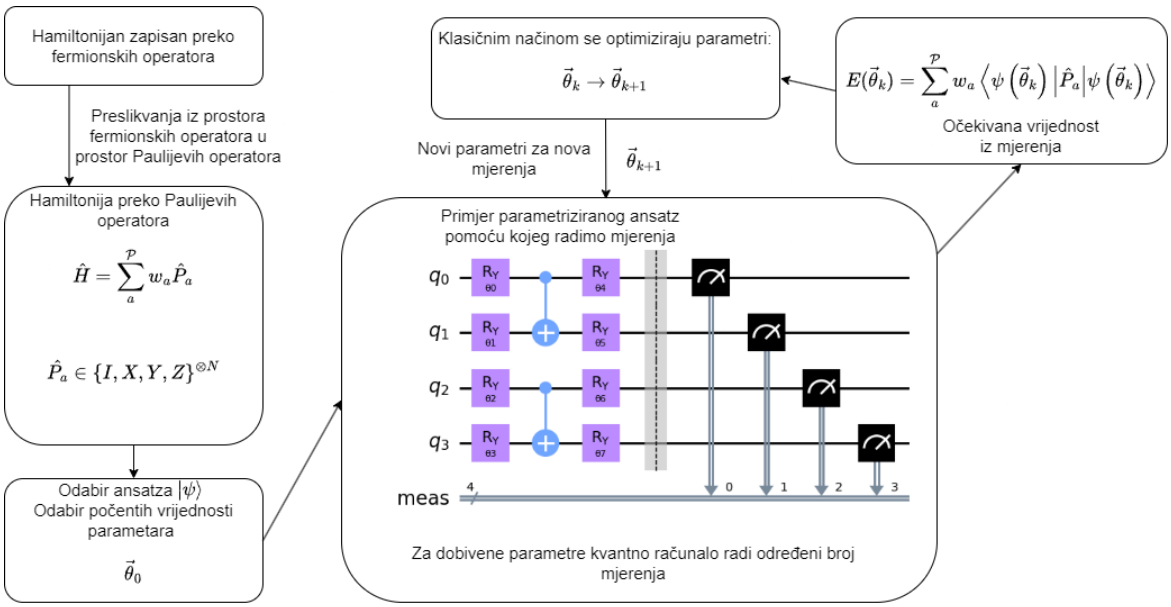
izvršava se na kvantnom računalu, dok se sumacija i minimizacija

$$E_{VQE} = \min_{\theta} \sum_{a} w_a E_{P_a} \tag{4.10}$$

izvršava na klasičnom računalu.

4.2 Dijelovi VQE-a

Sada ukratko prođimo kroz svaki dio algoritma. U sljedećim poglavljima će svaki korak biti detaljnije objašnjen. Promotrimo prikaz sheme VQE algoritma sa slike 16:



Slika 16: *Prikaza sheme VQE algoritma. Krenemo od hamiltonijana koji je zapisan preko fermionskih operatora, zatim taj hamiltonijan preslikamo u prostor Paulijevih operatora. Sljedeći korak je odabir nekog ansatza odnosno probne funkcije, ovaj odabir ovisi o početnom problemu. Za taj anastz još biramo početne parametre. Konstruiramo ansatz pomoću kvantnog sklopovlja i radimo definirani broj mjerenja za početne parametre. Iz tih mjerenja dobivamo očekivanu vrijednost energije. Tu vrijednost šaljemo klasičnom optimizatoru koji dobiva neke nove vrijednosti za parametre. Radimo mjerenja na kvantom računalu s tim novim parametrima. Ponavljamo ovo sve dok ne zadovoljimo neki postavljeni uvjet, primjerice dosegnut maksimalan broj evaluacija ili dosegnut postavljeni uvjeti konvergencije.*

1. **Prikaz hamiltonijana:**

   - Počinjemo s hamiltonijanom koji je zapisan preko fermionskih operatora u formalizmu koji se naziva druga kvantizacija. Više o ovom zapisu hamiltonijana će biti kasnije.
   - Ovaj hamiltonijan se preslikava u prostor Paulijevih operatora koristeći sljedeću for-

mulu:

$$\hat{H} = \sum_{a}^{\mathcal{P}} w_a \hat{P}_a$$

gdje je:

$$\hat{P}_a \in \{I, X, Y, Z\}^{\otimes N}$$

2. **Odabir ansatza:**

   - Sljedeći korak je odabir ansatza, odnosno probne funkcije. Ovaj odabir ovisi o početnom problemu. Ansatz je dan sa $|\psi\rangle$

3. **Odabir početnih vrijednosti parametara:**

   - Za odabrani ansatz, biramo početne parametre $\vec{\theta}_0$:

4. **Konstrukcija ansatza:**

   - Konstruiramo ansatz pomoću kvantnog sklopovlja i postavljamo odabrane početne parametre.

5. **Izvršavanje mjerenja:**

   - Radimo neki definirani broj mjerenja za trenutačne parametre.
   - Iz tih mjerenja, izračunavamo očekivanu vrijednost energije koristeći:

$$E(\vec{\theta}_k) = \sum_{a}^{\mathcal{P}} w_a \left\langle \psi\left(\vec{\theta}_k\right) \middle| \hat{P}_a \middle| \psi\left(\vec{\theta}_k\right) \right\rangle$$

6. **Optimizacija parametara i novo mjerenje:**

   - Očekivana vrijednost energije šalje se klasičnom optimizatoru.
   - Optimizator ažurira parametre $\boldsymbol{\theta}$ ansatza kako bi minimizirao očekivanu vrijednost $E(\boldsymbol{\theta_k})$ kvantnog hamiltonijana. Ključna komponenta većine optimizatora je izračun gradijenta koji je definiran kao:

$$\nabla E(\boldsymbol{\theta}) = \left(\frac{\partial E}{\partial \theta_1}, \frac{\partial E}{\partial \theta_2}, \ldots, \frac{\partial E}{\partial \theta_n} \right).$$

     Gradijent pokazuje kako mala promjena u parametru utječe na $E(\boldsymbol{\theta_k})$. Optimizatori koriste gradijent kako bi efikasno ažurirali parametre prema minimalnoj očekivanoj vrijednosti.

   - Radimo mjerenja na kvantnom računalu s novim parametrima.

7. **Konvergencija:**

- Ponavljamo korake 4, 5, i 6 sve dok ne zadovoljimo neki postavljeni uvjet.
- Uvjeti mogu biti, primjerice, dosegnut maksimalan broj evaluacija ili postavljeni uvjeti konvergencije.

4.3 Reprezentacija hamiltonijana

Pođimo prvo od konstrukcije hamiltonijan za ab initio molekulu. Neka promatramo ne relativistička svojstva i neka vrijedi Born-Oppenheimerova aproksimacija, gdje se zanemaruje gibanje jezgre. Tada hamiltonijan možemo pisati na sljedeći način:

$$\hat{H} = \hat{T}_e + \hat{V}_{ne} + \hat{V}_{ee} \tag{4.11}$$

gdje su kinetička energije elektrona $\hat{T_e}$, potencijalna energija interakcije elektrona i jezgre $\hat{V_{ne}}$ te potencijalan energija interakcije između elektrona $\hat{V_{ee}}$ dana s

$$\begin{aligned} \hat{T}_e &= -\sum_i \frac{\hbar^2}{2M_i}\nabla_i^2, \\ \hat{V}_{ne} &= -\sum_{i,k} \frac{e^2}{4\pi\epsilon_0}\frac{Z_k}{|\mathbf{r}_i - \mathbf{R}_k|}, \\ \hat{V}_{ee} &= \frac{1}{2}\sum_{i\neq j} \frac{e^2}{4\pi\epsilon_0}\frac{1}{|\mathbf{r}_i - \mathbf{r}_j|}, \end{aligned} \tag{4.12}$$

gdje $\mathbf{r}_i$ predstavlja poziciju i-tog elektrona, M_i masu, Z_k je atomski broj jezgre k i e jest elementarni naboj. S obzirom da su elektroni fermioni ukupna valna funkcija mora biti antisimetrična. Ovo se zgodno može napisati preko Slaterove determinante

$$\psi(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n) = \frac{1}{\sqrt{n!}} \begin{vmatrix} \phi_1(\mathbf{x}_1) & \phi_2(\mathbf{x}_1) & \cdots & \phi_n(\mathbf{x}_1) \\ \phi_1(\mathbf{x}_2) & \phi_2(\mathbf{x}_2) & \cdots & \phi_n(\mathbf{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(\mathbf{x}_n) & \phi_2(\mathbf{x}_n) & \cdots & \phi_n(\mathbf{x}_n) \end{vmatrix} \tag{4.13}$$

ovdje $\phi_j(\mathbf{x}_j)$ predstavlja valnu funkciju za spinski i orbitalni dio pojedinog elektrona, gdje je $\mathbf{x}_j = (\mathbf{r}_j, \sigma_j)$. Cijela funkcija stanja baze za n elektrona se može kraće zapisati na sljedeći način

$$|\psi\rangle = |\phi_1\phi_2\ldots\phi_n\rangle \tag{4.14}$$

gdje stavljamo $\phi_i = 1$ ako je j-ta pozicija okupirana, a 0 ako nije. Reprezentaciju na ovaj način možemo jednostavno prikazati preko qubita. Postavlja se pitanje kako uvjet antisimetričnosti zadovoljiti za valnu funkciju. To se može postići na dva načina, da se prilikom konstrukcije valne funkcije zadovolji uvjet antisimetričnosti ili da se antisimetričnosti nametne putem ko-

mutacijskih relacija. Prvi način poznat je kao prva kvantizacija, češće korišten u kvantnoj mehanici [3]. Drugi način nazivamo druga kvantizacija.

4.3.1 Prva kvantizacija

Kako se ovdje antisimetričnost uvodi u konstrukciji valne funkcije, hamiltonijan se može konstruirati kao projekcija na bazu jedne čestice

$$\hat{H} = \sum_{i=1}^{m} \sum_{p,q=1}^{n} h_{pq} \left|\phi_p^{(i)}\right\rangle \left\langle \phi_q^{(i)} \right| + \frac{1}{2} \sum_{i \neq j}^{m} \sum_{p,q,r,s=1}^{n} h_{pqrs} \left| \phi_p^{(i)} \phi_q^{(j)} \right\rangle \left\langle \phi_r^{(i)} \phi_s^{(j)} \right| \tag{4.15}$$

gdje matrični element za jedan elektron jest

$$\begin{aligned} h_{pq} &= \left\langle \phi_p \left| \hat{T}_e + \hat{V}_{ne} \right| \phi_q \right\rangle \\ &= \int \mathrm{d}\mathbf{x} \phi_p^*(\mathbf{x}) \left(-\frac{\hbar^2}{2m_e} \nabla^2 - \sum_k \frac{e^2}{4\pi\epsilon_0} \frac{Z_k}{|\mathbf{r} - \mathbf{R}_k|} \right) \phi_q(\mathbf{x}). \end{aligned} \tag{4.16}$$

Za interakciju dva elektron imamo

$$\begin{aligned} h_{pqrs} &= \left\langle \phi_p \phi_q \left| \hat{V}_{ee} \right| \phi_r \phi_s \right\rangle \\ &= \frac{e^2}{4\pi\epsilon_0} \int \mathrm{d}\mathbf{x}_1 \, \mathrm{d}\mathbf{x}_2 \frac{\phi_p^*(\mathbf{x}_1) \, \phi_q^*(\mathbf{x}_2) \, \phi_r(\mathbf{x}_2) \, \phi_s(\mathbf{x}_1)}{|\mathbf{r}_1 - \mathbf{r}_2|}. \end{aligned} \tag{4.17}$$

Postoji mogućnost preslikavanja ovih n jedno-čestičnih funkcija baze u binarne brojeve pa potom u qubite [7]. Ova metoda preko prve kvantizacije mnogo je zahtjevnija pa se ne koristi, već se problemu pristupa preko druge kvantizacije [3].

4.3.2 Druga kvantizacija

Druga kvantizacija kroz konstrukciju operatora zadovoljava antisimetričnost. Stoga operatori zadovoljavaju neke dodatne uvjete uz to što djelovanjem moraju omogućiti čestici pomicanje iz jedne orbitale u drugu. Ovi operatori se nazivaju fermionski operatori stvaranja i poništavanja. Fizičari Jordan i Wigner uveli su kanonske fermionske antikomutacijske relacije iz koji se mogu dobiti komutacijske relacije [8]. Komutator dvaju operatora $\hat{A}$ i $\hat{B}$ definiran je kao $\left[\hat{A}, \hat{B}\right] = \hat{A}\hat{B} - \hat{B}\hat{A}$, a antikomutator dvaju operatora $\hat{A}$ i $\hat{B}$ kao $\left\{\hat{A}, \hat{B}\right\} = \hat{A}\hat{B} + \hat{B}\hat{A}$. Za fermionske operatore se onda dobije

$$\begin{aligned} \left\{\hat{a}_p, \hat{a}_q^\dagger\right\} &= \delta_{pq} \\ \left\{\hat{a}_p^\dagger, \hat{a}_q^\dagger\right\} &= \{\hat{a}_p, \hat{a}_q\} = 0 \end{aligned} \tag{4.18}$$

$$\begin{aligned}[\hat{a}_p, \hat{a}_q] &= -2\hat{a}_q\hat{a}_p, \\ \left[\hat{a}_p, \hat{a}_q^\dagger\right] &= \delta_{pq} - 2\hat{a}_q\hat{a}_p\end{aligned} \tag{4.19}$$

Slaterova determinanta se može napisati na sljedeći način

$$|\psi\rangle = \prod_i \left(\hat{a}_i^\dagger\right)^{\phi_i} |\text{vac}\rangle = \left(\hat{a}_1^\dagger\right)^{\phi_1} \left(\hat{a}_2^\dagger\right)^{\phi_2} \cdots \left(\hat{a}_n^\dagger\right)^{\phi_n} |\text{vac}\rangle \tag{4.20}$$

gdje je stanje $|\text{vac}\rangle$, stanje vakuuma koje nestane kada na njega djelujemo operatorom poništavanja. Promotrimo sada kako operatori djeluju na stanje $|\phi_1\phi_2\cdots\rangle$.

$$\begin{aligned}\hat{a}_j^\dagger |\phi_1\phi_2\cdots\rangle &= \begin{cases} 0, & \phi_j = 1 \\ s_p |\phi_1\phi_2\cdots 1_j \cdots\rangle, & \phi_j = 0 \end{cases} \\ \hat{a}_j |\phi_1\phi_2\cdots\rangle &= \begin{cases} 0, & \phi_j = 0 \\ s_p |\phi_1\phi_2\cdots 0_j \cdots\rangle, & \phi_j = 1 \end{cases}\end{aligned} \tag{4.21}$$

Odnosno, operator stvaranja $\hat{a}_j^\dagger$ rezultira nulom ako je stanje već zauzeto, ako nije onda zauzima to stanje. Operator poništavanja $\hat{a}_j$ uklanja fermion iz orbitale, ako je ta orbitala zauzeta. Još imamo s_p koji predstavlja parnost p-te orbitale. Točnije, s_p je 1 ili -1 ovisno o tome je li broj zauzetih orbitala sve do (ali ne uključujući) p-tu orbitalu paran ili neparan.
Hamiltonijan iz jednadžbe (4.15) se može napisati preko fermionskih operatora, izvod je podosta dug i ovdje neće biti prikazan [9], dobije se

$$\hat{H} = \sum_{pq} h_{pq}\hat{a}_p^\dagger\hat{a}_q + \frac{1}{2}\sum_{pqrs} h_{pqrs}\hat{a}_p^\dagger\hat{a}_q^\dagger\hat{a}_r\hat{a}_s. \tag{4.22}$$

4.4 Iz fermionskog prostora u spinski prostor

Kako već znamo qubiti su spin 1/2 objekti i stoga su Paulijevi operatori X, Y i Z jedini koje možemo direktno mjeriti na kvantnim računalima. Međutim, oni ne poštuju iste relacije kao fermionski operatori stvaranja i poništavanja. Stoga je potrebno fermionske operatore napisati preko spinskih kako bi hamiltnoijan (4.22) mogli primijeniti na kvantnim računalima. Jordan i Wigner su još i prije kvantnih računala pokazali da postoji izomorfizam između ova dva prostora [8].
Prilikom odabira načina preslikavanja treba uzeti u obzir tri faktora [3]:

- *Broj qubita*: Ovaj faktor je jasan sam po sebi. S obzirom da je hardverski teško implementirati veliki broj qubita, preslikavanja koja u konačnici daju manje qubita su poželjnije.
- *Paulijevu težinu*: Odnosi se na broj operatora koji su različiti od jediničnog za svaki

tenzorski produkt.

- *Broj tenzorskih produkta Paulijevih operatora*: Ovo je još jedan važan faktor koji treba uzeti u obzir prilikom odabira metode preslikavanja.

Spomenuti ćemo generalna preslikavanja koja preslikavaju cijeli Fockov prostor za neki hamiltonijan. Ova vrsta preslikavanja se koriste za molekule dok se za modele rešetke koriste druga vrsta koja više lokalno preslikava iz jednog prostora u drugi [3].

4.4.1 Jordan-Wignerovo preslikavanje

Jordan-Wignerovo preslikavanje preslikava fermionsku valnu funkciju u skup qubita tako da preslika okupacijski broj spinskih orbitala u stanja qubita. Stanje $|0\rangle_j$ govori da j-ta spinska orbitala nije okupirana, kada imamo jedinicu onda je okupirana. Također znamo da $a_j^\dagger|0\rangle_j = |1\rangle_j$ i $a_j^\dagger|1\rangle_j = 0$, pa se jednostavno pokaže da vrijedi

$$\begin{aligned} a_j^\dagger &= \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} = \frac{X_j - iY_j}{2} \\ a_j &= \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} = \frac{X_j + iY_j}{2} \end{aligned} \tag{4.23}$$

gdje spinski operatori X i Y djeluju na j-ti qubit. Ovo je primjer za jednu orbitalu, no kada imamo dvije ili više orbitala nailazimo na problem. Fermioni radi antisimetričnosti zadovoljavaju relaciju $a_j^\dagger a_k^\dagger = -a_k^\dagger a_j^\dagger$ za bilo koji j i k, ako ovo zapišemo preko relacija iz (5.23) dolazimo do problema

$$\left(\frac{X_j - iY_j}{2}\right)\left(\frac{X_k - iY_k}{2}\right) = \left(\frac{X_k - iY_k}{2}\right)\left(\frac{X_j - iY_j}{2}\right). \tag{4.24}$$

Odnosno operatori stvaranja ne antikomutiraju kako bi trebali. Kako bi se ovo ispravilo uzme se u obzir da Paulijevi operatori antikomutiraju, $XZ = -ZX$ i $YZ = -ZY$. Stoga ubacivanjem operatora Z u konstrukciju dolazimo do željene antikomutacijske relacije. Konstrukcija je

prikazana izrazima

$$\begin{aligned} a_1^\dagger &= \left(\frac{X-iY}{2}\right) \otimes 1 \otimes 1 \otimes 1 \otimes \cdots \otimes 1 \\ a_2^\dagger &= Z \otimes \left(\frac{X-iY}{2}\right) \otimes 1 \otimes 1 \otimes \cdots \otimes 1 \\ a_3^\dagger &= Z \otimes Z \otimes \left(\frac{X-iY}{2}\right) \otimes 1 \otimes \cdots \otimes 1 \\ &\vdots \\ a_N^\dagger &= Z \otimes Z \otimes Z \otimes Z \otimes \cdots \otimes Z \otimes \left(\frac{X-iY}{2}\right) \end{aligned} \tag{4.25}$$

Još je zgodno operator broja čestica n_j, koji je definiran kao $a_j^\dagger a_j$, prikazati preko Paulijevih operatora. Ovaj operator broji koliko fermiona se nalazi na j-toj poziciji. Kada se raspiše umnožak $a_j^\dagger a_j$, nizovi Z operatora se ponište jer vrijedi $Z^2 = 1$. Tako dolazimo do izraza $n_j = \left(\frac{X_j - iY_j}{2}\right)\left(\frac{X_j + iY_j}{2}\right)$. Još iz kvante fizike je poznato da vrijedi $X_jY_j = iZ_j$ pa slijedi relacija

$$n_j = \frac{X_j^2 + Y_j^2}{4} - \frac{i\left(X_jY_j - Y_jX_j\right)}{4}. \tag{4.26}$$

Primijetimo još da vrijedi $X_j^2 = Y_j^2 = 1$ i $X_jY_j - Y_jX_j = 2iZ_j$. Konačno dolazimo do izraza

$$n_j = \frac{1 - Z_j}{2}. \tag{4.27}$$

Jedna od posljedica dodavanja tenzorskih produkta Z operatora je da Jordan-Wigner preslikavanje postaje poprilično skupo [10]. Odnosno Paulijeva težina skalira s $\mathcal{O}(N)$. Unatoč tome se koristi jer je jedan od najjednostavnijih pristupa problemu preslikavanja iz fermionskog prostora u spinski [3].

4.4.2 Paritetno preslikavanje

Umjesto da koristimo j-ti qubit za informaciju o zauzetosti j-te orbitale, kod ovog preslikavanja j-ti qubit govori o parnosti orbitale. Podsjetimo se, parnost govori o broj zauzetih orbitala sve do j-te orbitale. Ako je broj zauzetih orbitala sve do j-te i uključujući nju paran, j-ti qubit je u stanju $|0\rangle$. Ako je pak broj neparan onda je j-ti qubit u stanju $|1\rangle$.
Dalje ako imamo fermionsko stanje $|v_0v_1 \ldots v_n\rangle$ možemo ga preslikati u qubitno stanje $|p_0p_1 \ldots p_n\rangle$ pomoću sljedeće transformacije

$$\begin{aligned} p_i &= \textstyle\sum_{j \le i} v_j \qquad (\mathrm{mod} 2) \\ &= \textstyle\sum_j \left[\pi_n\right]_{ij} v_j \quad (\mathrm{mod} 2), \end{aligned} \tag{4.28}$$

odnosno

$$|\boldsymbol{p}\rangle = |\pi_n(\boldsymbol{v})\rangle \quad (\mathrm{mod} 2) \tag{4.29}$$

gdje se uzima modul 2 kako bi dobili parnost, a ne sumu. Još treba definirati matricu π_n koja je $n \times n$ matrica definirana na sljedeći način

$$\pi_n = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 1 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \cdots & 1 \end{bmatrix} \tag{4.30}$$

Promjena parnosti na j-toj poziciji ukazuje na zauzetost j-te orbitale. Općenito, ako se parnost promijeni između $j-1$-te i j-te pozicije (npr. $|0\rangle_{j-1}$ i $|1\rangle_j$ ili obrnuto), to znači da je j-ta orbitala zauzeta. Ako parnost ostane ista (npr. oba qubita su $|0\rangle$ ili oba su $|1\rangle$), to znači da je j-ta orbitala prazna.
U ovoj reprezentaciji imamo informaciju o parnosti qubita sve do j-tog qubita u prethodnom $(j-1)$ qubitu. Iz (4.21) se onda lako vidi da fermionski operatori $\hat{a}_j^\dagger$ i $\hat{a}_j$ daju minus znak kada imamo $|1\rangle_{j-1}$ te plus kada je $|0\rangle_{j-1}$. Uzimajući u obzir qubite samo do i uključujući j-ti qubit preslikavanje bi definirali na sljedeći način

$$\begin{aligned} \hat{a}_j^\dagger &\to \frac{Z_{j-1} \otimes (X_j - iY_j)}{2} \\ \hat{a}_j &\to \frac{Z_{j-1} \otimes (X_j + iY_j)}{2} \end{aligned} \tag{4.31}$$

Kada dodajemo ili uklanjamo elektron iz j-te orbitale koristeći fermionske operatore $\hat{a}_j^\dagger$ i $\hat{a}_j$, mijenjamo parnost qubita koji slijede nakon j-tog qubita. Da bismo ažurirali parnost, koristimo niz X operatora koji djeluju na qubite koji slijede nakon j-tog qubita. Ovo je ključno za održavanje ispravne parnosti u cijelom sustavu nakon što se promijeni zauzetost j-te orbitale. Uzimajući i ovo u obzir dobivamo potpunu transformaciju

$$\begin{aligned} \hat{a}_j^\dagger &\to \frac{Z_{j-1} \otimes (X_j - iY_j)}{2} \otimes X_{j+1} \otimes \cdots \otimes X_{n-1} \\ \hat{a}_j &\to \frac{Z_{j-1} \otimes (X_j + iY_j)}{2} \otimes X_{j+1} \otimes \cdots \otimes X_{n-1} \end{aligned} \tag{4.32}$$

4.5 Odabir ansatza

Odabir ansatza predstavlja ključni korak u VQE postupku. Pravilnim izborom ansatza možemo se približiti stvarnom željenom stanju. Glavni cilj je proširiti raspon ansatza u onim dijelovima Hilbertovog prostora koji sadrže rješenje. Raspon stanja koja ansatz može doseći nazivamo *izražajnost*. Međutim, postoji izazov: ansatz s visokom izražajnošću može biti težak za optimizaciju zbog velikog broja parametara. Ključno pitanje je može li se ansatz optimizirati na

praktičan način, što se odnosi na njegovu *obučljivost*. U stvarnom svijetu, često je bolje odabrati ansatz koji pokriva manji dio prostora, ali je lako obučljiv. Dakle, izazov je pronaći ansatz koji postiže ravnotežu između izražajnosti i obučljivosti.

Obučljivost se odnosi na sposobnost efikasnog pronalaženja optimalnog skupa parametara ansatza putem iterativne optimizacije [11]. Obučljivost ansatza može se procijeniti pomoću ponašanja njegovog gradijenta tijekom optimizacije. Ako gradijent polinomski nestaje, ansatz je obučljiv jer se parametri mogu prilagoditi kako bi se postigla željena preciznost. Međutim, ako gradijent eksponencijalno nestaje, suočavamo se s problemom "neplodne visoravni" (engl. Barren Plateau), što znači da optimizacija može postati neizvediva [12]. U sljedećem poglavlju biti će više o ovome problemu.

4.5.1 Problem "neplodne visoravni"

Ovaj problem predstavlja jednu od potencijalno velikih prepreka u daljnjem razvoju VQE algoritma [3]. On nastaje kada gradijent funkcije troška eksponencijalno nestaje, što znači da optimizacija postaje izuzetno teška i može se zaustaviti u određenim dijelovima prostora parametara. U takvim situacijama, optimizatori ne mogu pronaći smjer u kojem bi trebali ažurirati parametre kako bi smanjili funkciju troška.
Podsjetimo se, funkcija troška je definirana s

$$E_{\mathrm{VQE}} = \min_{\boldsymbol{\theta}} \left\langle \mathbf{0} \left| U^{\dagger}(\boldsymbol{\theta}) \hat{H} U(\boldsymbol{\theta}) \right| \mathbf{0} \right\rangle .$$

gdje je E_{VQE} očekivana vrijednost hamiltonijana, $U(\boldsymbol{\theta})$ je parametrizirani unitarni operator, a $\hat{H}$ je hamiltonijan koji želimo analizirati.
Kada gradijent ove funkcije troška eksponencijalno nestane, to znači da promjene u parametrima $\boldsymbol{\theta}$ ne rezultiraju značajnim promjenama u E_{VQE}. Kao rezultat toga, optimizatori ne mogu lako odrediti kako ažurirati parametre.
Za razumijevanje ovog problem zgodna je analogija da se spuštamo niz planinu i želimo doći do dna. U slučaju polinomskog nestajanja, padina planine postupno se smanjuje kako se spuštamo i lako nalazimo dno. S druge strane, u slučaju eksponencijalnog nestajanja, nakon samo nekoliko koraka, osjećamo kao da hodamo po gotovo ravnoj površini, čak i ako nismo blizu dna.
Uzroku ovome problemu mogu biti klasični problemi koji se javljaju i u strojnom učenju, a to su loša inicijalizacija početnih parametara te prevelika izražajnost, koja ima inverzni odnos s obučljivosti [12]. Uzroci kvantne prirode se javljaju jer su procjene na kvantom računalu stohastičke prirode, odnosno svaka opservabla se može mjeriti do neke preciznosti, povećavajući tu preciznost s brojem mjerenja. Kada gradijent funkcije troška ide prema nuli, postaje teže razlikovati između pozitivnog i negativnog gradijenta čime optimizacija postaje nasumičan hod. Još treba spomenuti da kod strojnog učenja vrijedi: što imamo više razina to problem postaje

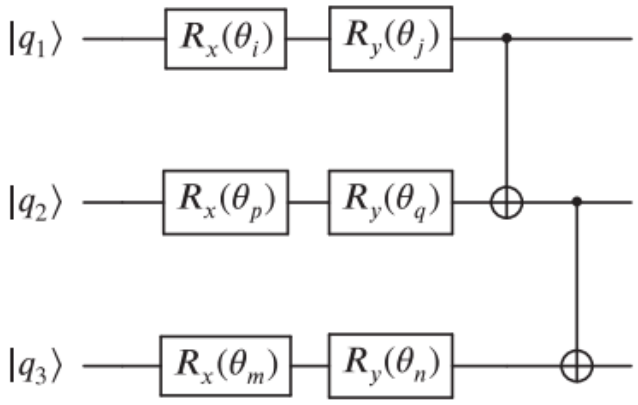
teži. Kod kvantnih računala veći broj qubita trebao bi voditi na poboljšanje. Međutim, hardver današnjih kvantnih računala još uvijek ne omogućava veliki broj qubita, tako da je čest uzrok problema "neplodne visoravni" [3].
Ovaj problem je jedan od najznačajnijih prepreka za rad VQE algoritma. Nije još jasno hoće li se uspješno riješiti i omogućiti izvršavanje VQE-a na današnjim NISQ uređajima [3].

4.5.2 Fiksni ansatzi

Fiksni ansatzi su oni koji se inicijaliziraju na početku i ostaju fiksni tijekom cijelog procesa optimizacije, naravno osim vrijednosti parametara koji se ažuriraju. Postoje i adaptivni ansatzi gdje se i struktura ansatza mijenja [3], no u ovom radu biti će spomenuti samo fiksni.

4.5.3 Hardverski efikasni ansatzi (HEA)

Ova vrsta ansatza specifično je dizajnirana da radi što efikasnije na kvantom računalu. To se postiže na način da se probno stanje VQE parametrizira kvantnim operatorima izravno prilagođenim kvantnom uređaju na kojem se eksperiment izvodi [3]. Ključna karakteristika ove vrste ansatza je jednostavnost.
Konstrukcija hardverski efikasnih ansatza se obično implementira tako da se ponavljaju blokovi unarnih, parametriziranih i rotacijskih operatora te operatora koji stvaraju spregnuta stanja. Parametrizirani operatori su operatori koji sadrže parametre koje možemo prilagoditi tijekom optimizacije. Na primjer, među najčešće korištenim parametriziranim operatorima su rotacijski operatori $R_y(\theta)$ i $R_x(\theta)$, gdje je θ parametar koji određuje kut rotacije. Tijekom optimizacije VQE algoritma, ovaj kut θ se prilagođava kako bi se minimizirala očekivana vrijednost hamiltonijana. Hardverski efikasni ansatzi su dizajnirani za optimalno izvođenje na kvantnim računalima. Međutim, zbog te općenitosti, često pokrivaju širi dio Hilbertovog prostora nego što



Slika 17: *Primjer hea gdje R_x i R_y djeluju kao rotacijski operatori, a potom $CNOT$ operator djeluje na dva qubita kako bi dobili spregnuto stanje [3].*

je potrebno za određeni problem [3]. Iako su prilagođeni kvantnim računalima, nisu nužno najučinkovitiji za svaki problem. Radi ovoga HEA odmah postaje izložena problemu "neplodne visoravni". Također ova vrsta ansatza zahtijeva mnogo više parametara nego neka druga, primjerice za implementaciju VQE-a za H_2O preko hardverski efikasnih ansatza je trebalo 198 parametara, a za UCCSD vrstu ansatza, koju ćemo kasnije uvesti, samo 8 parametara [13]. Kako stvari za sada stoje HEA nije vrsta ansatza koja će biti praktična za simulaciju kompliciranijih molekula [3].

4.5.4 UCC ansatz

Ova vrsta ansatza dolazi iz metode spregnutog klastera (Coupled Cluster CC) koja se kvantnoj kemiji koristi za dobivanje elektronskih struktura i energija. Kreće od osnovne Hartree-Fock metode pa dalje razvija tu valnu funkciju za detaljnije rezultate [14].
U CC teoriji je glavna ideja da se valna funkcija sustava s više čestica prikaže kao evolucija inicijalne Hartree-Fock valne funkcije

$$|\psi\rangle = e^{\hat{T}} |\psi_{HF}\rangle \tag{4.33}$$

gdje je $\hat{T}$ klaster operator, definiran s

$$\hat{T} = \hat{T}_1 + \hat{T}_2 + \cdots \hat{T}_v \tag{4.34}$$

gdje v označava maksimalno dopušteno pobuđeno stanje. Operator $\hat{T}_1$ je operator za prvu pobuđeno stanje, $\hat{T}_2$ za drugo, itd. Dakle, osnovna ideja CC teorije je da, umjesto da izravno rješavamo vrlo složeni problem, možemo početi s nekim početnim približnim rješenjem (Hartree-Fock) i zatim klaster operatorima pobuditi inicijalno stanje kako bi došli točnijeg rješenja.
U formalizmu druge kvantizacije primjera za prva dva operatora je

$$\begin{aligned} T_1 &= \sum_i \sum_a t_a^i \hat{a}^a \hat{a}_i, \\ T_2 &= \frac{1}{4} \sum_{i,j} \sum_{a,b} t_{ab}^{ij} \hat{a}^a \hat{a}^b \hat{a}_j \hat{a}_i, \end{aligned} \tag{4.35}$$

gdje operatori poništavanja djeluju na orbitale koje su zauzete u HF determinanti (i, j), a operatori stvaranja u HF determinantu uključuju jednu od prethodno nezauzetnih orbitala (tzv. virtualnih orbitala), opisanih indeksima (a, b). t_a^i i t_{ab}^{ij} su klaster koeficijenti čije se vrijednosti mogu odrediti iz elektronske Schrödingerove jednadžbe [15].
Kako operator $e^{\hat{T}}$ nije unitaran, implementacija na kvantom računalu nije moguća, stoga se razvila metoda unitarnog spojenog klaster (unitary coupled cluster UCC) gdje se rješava ovaj problem. Činjenica da za bilo koji linearni operator $\hat{T}$ vrijedi da je $\left(\hat{T} - \hat{T}^\dagger\right)$ antihermitski, a eksponencijalne funkcija antihermitskog operatora rezultira u unitarnom operatoru [3]. Stoga

za UCC vrijedi

$$|\psi\rangle = e^{\hat{T}-\hat{T}^\dagger} |\psi_{HF}\rangle \,. \tag{4.36}$$

U klasičnim računalima ovaj pristup skalira eksponencijalno pa ima smisla pokušati s kvantnim računalima [3]. Kako se ovdje radi o VQE algoritmu treba još pokazati kako doći do parametrzirane verzije UCC-a, to činimo na način da grupiramo individualne operatore pobuđenja $\hat{T}$ s njihovim konjugatom $\hat{T}^\dagger$ i za njih uvedemo oznaku τ. Ovime dolazimo do parametrizirane verzije UCC ansatza.

$$U(\vec{t}) = e^{\sum_j t_j (\tau_j - \tau_j^\dagger)} \tag{4.37}$$

gdje t_j prikazuje pojedine amplitude koje su se javljale u jednadžbi 5.35. Konačno još dobiveni parametrizirani ansatz korištenjem prethodni opisanih fermionskih operatora prebacujemo u Paulijeve operatora kako bi omogućili izvođenje na kvantom računalu. U ovom radu fokus će biti na UCSSD ansatzu. Ovaj ansatz kombinira operatore $\hat{T}^1$ i $\hat{T}^2$ koji predstavljaju prvo i drugo pobuđeno stanje. Kroz ovu kombinaciju, ansatz može modelirati i pojedinačne i dvostruke pobude elektrona. UCCSD ansatz se pokazao kao jedan od boljih za probleme kvante kemije [3].

4.6 Optimizacija

Cijeli proces VQE-a u konačnici se svodi na optimizaciju. Odabir optimizatora ključan za uspješnost VQE algoritma. Iako se mnoge tehnike optimizacije preuzimaju iz područja strojnog učenja, priroda kvantnih računala donosi dodatne izazove. Prva prepreka su šumovi koji se javljaju na kvantnim računalima. Budući da smo još uvijek u eri NISQ kvantnih računala, mnogi optimizatori su osjetljivi na šumove, stoga je važno pažljivo odabrati koji optimizator koristiti [16].
Druga prepreka je preciznost aproksimiranih vrijednosti. Aproksimaciju radimo temeljem mjerenih veličina na kvantnom računalu. Stoga, optimizacija može biti neprecizna ako je sama aproksimacija netočna zbog malog broja mjerenja [3].
Treća prepreka je problem "niske visoravni", koji je bio diskutiran u sekciji 4.6.1.

4.6.1 Formalnost optimizacije u VQE-u

VQE algoritam izbaci funkciju koja se dobije iz rezultata mjerenja. Ovu funkciju nazivamo objektivnom funkcijom, to je nomenklatura naslijeđena iz računarstva. U našem slučaju funkciju označujemo s $\mathcal{L}(\boldsymbol{\theta})$, gdje $\boldsymbol{\theta}$ prikazuje parametre kvantnog kruga. Funkciju konstruiramo iz

opservabli, odnosno kvantnih operatora koje mjerimo

$$\mathbf{O}(\boldsymbol{\theta}) = \left(\hat{O}_1\left(\boldsymbol{\theta}^{(1)}\right), \hat{O}_2^{(2)}\left(\boldsymbol{\theta}^{(2)}\right), \ldots, \hat{O}_a\left(\boldsymbol{\theta}^{(a)}\right)\right). \tag{4.38}$$

Imamo još funkciju C koja preslikava izmjerene vrijednosti u objektivnu funkciju. Funkcija C je obično linearna

$$C(\mathbf{X}) = \sum_i c_i X_i, \tag{4.39}$$

gdje su c_i konstante, a X_i su očekivane vrijednosti observabli. Pa za objektivnu funkciju slijedi [3]

$$\mathcal{L}(\boldsymbol{\theta}) = C(\mathbf{O}(\boldsymbol{\theta})). \tag{4.40}$$

Povežimo ovo još s dijagramom sa slike 16. Tu je objektivna funkcija

$$E\left(\vec{\theta}_k\right) = \sum_a^{\mathcal{P}} w_a \left\langle \psi\left(\vec{\theta}_k\right) \middle| \hat{P}_a \middle| \psi\left(\vec{\theta}_k\right) \right\rangle.$$

Očekivane vrijednosti X_a su dane s $\left\langle \psi\left(\vec{\theta}_k\right) \middle| \hat{P}_a \middle| \psi\left(\vec{\theta}_k\right) \right\rangle$ gdje $\hat{P}_a$ označava Paulijeve operatore unutar nekog tenzorskog produkta.
Navesti ćemo generalne klase optimizatora i spomenuti one koje koristimo kasnije u radu.

4.6.2 Optimizatori koji evaluiraju gradijent

Optimizatori koji direktno računaju gradijent: Ovi optimizatori koriste derivacije funkcije troška kako bi pronašli optimalno rješenje. Uvodimo pojam reda optimizatora koji zapravo samo odgovara redu derivacije. Odnosno optimizatori prvog reda koriste samo prvu derivaciju dok optimizatori drugog reda koriste prvu i drugu derivaciju [3]. U ovom radu koristiti ćemo tri vrste ovih optimizatora. Kako ovi optimizatori koriste napredne metode iz numeričke matematike bit će u kratko spomenuti i referencirani radovi za daljnje proučavanje.

- **TNC optimizator** koristi skraćeni Newtonov algoritam kako bi minimizirao funkcije koje su podložne granicama, ovaj algoritam koristi informacije o gradijentu [17].
- **SLSQP optimizator** je implementacija SQP metode u `scipy` biblioteci koja koristi iterativni pristup za nelinearnu optimizaciju s ograničenjima. SQP metode mogu se smatrati kvazi-Newtonovim metodama. Koriste se za matematičke probleme kod kojih su ciljna funkcija i ograničenja dva puta neprekidno diferencijabilna [18].
- **L_BFGS_B optimizator**, je optimizator čiji je cilj minimizirati vrijednost diferencijabilne skalarne funkcije f. Ovaj optimizator je kvazi-Newtonova metoda, što znači da, za razliku od Newtonove metode, ne zahtijeva Hessian od f (matricu drugih derivacija f) prilikom pokušaja izračuna minimalne vrijednosti f [19].

Optimizatori koji evaluiraju gradijent uz pomoć stohastičkih metoda: Ova vrsta optimizatora koristi stohastičke metode kako bi aproksimirala gradijent objektivne funkcije. Ovo čini problem jednostavnijim jer pojednostavljuje evaluaciju očekivanih vrijednosti unutar VQE algoritma [3].
Prođimo kroz SPSA optimizator koji je jedan od najotpornijih optimizatora na šumove. Razmotrimo funkciju troška $\mathcal{L}(\boldsymbol{\theta})$ gdje je $\boldsymbol{\theta}$ vektor dimenzije p. Optimizacijski problem može se prevesti u pronalaženje optimalnog skupa parametara $\boldsymbol{\theta}^*$ gdje je $\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} = 0$. SPSA započinje s početnim vektorom parametara $\hat{\boldsymbol{\theta}}_0$ i njegovo pravilo ažuriranja je:

$$\hat{\boldsymbol{\theta}}_{k+1} = \hat{\boldsymbol{\theta}}_k - a_k \hat{g}_k \left(\hat{\boldsymbol{\theta}}_k \right),$$

gdje je $\hat{g}_k$ stohastička procjena gradijenta u iteraciji $\hat{\boldsymbol{\theta}}_k$ temeljena na prethodnim mjerenjima funkcije troška, a a_k je faktor koji određuje veličinu koraka.
U SPSA, procijenjeni gradijent u svakom koraku iteracije izražen je kao:

$$\hat{g}_{ki} \left(\hat{\boldsymbol{\theta}}_k \right) = \frac{y \left(\hat{\boldsymbol{\theta}}_k + c_k \Delta_k \right) - y \left(\hat{\boldsymbol{\theta}}_k - c_k \Delta_k \right)}{2 c_k \Delta_{ki}},$$

gdje je c_k pozitivan broj, a Δ_k je vektor smetnji. Svaka komponenta vektora Δ_k označena s Δ_{ki} predstavlja smetnju za i-ti parametar. Nasumičnost tehnike dolazi od činjenice da se komponente vektora smetnji Δ_k nasumično generiraju u svakom koraku iteracije što i daje otpornost na šumove.

4.6.3 Optimizatori bez gradijenta

Postoje i razni optimizatori koji optimiziraju bez evaluacije gradijenta.
COBYLA (Constrained Optimization BY Linear Approximations) je metoda optimizacije koja umjesto gradijenta funkcije koristi linearnu aproksimaciju kako bi pronašla optimalno rješenje. Odnosno COBYLA konstruira uzastopne linearne aproksimacije ciljne funkcije i ograničenja koristeći simpleks od $n + 1$ točaka (u n dimenzija). Zatim u svakom koraku optimizira te aproksimacije unutar povjerenog područja. Ovu metodu je razvio Powell 1994. godine [20].

5 Implementacija VQE-a u Qiskitu

Nakon detaljne analize svakog koraka VQE-a, razmotrit ćemo njegovu implementaciju kroz različite primjere u Qiskitu. Tijekom ovog rada koristit ćemo tri metode za izvršavanje VQE-a. Svaka od ovih metoda se oslanja na klasu `Estimator` u Qiskitu [21], čija će implementacija biti prikazana kasnije u kodu.

1. **Egzaktna simulacija**: Egzaktne očekivane vrijednosti dobivamo kada je argument `shots` klase `Estimator` postavljen postavljen na `None`, `Estimator` tada izračunava očekivanu vrijednost $\langle O \rangle = \langle \psi | O | \psi \rangle$. Ovdje $\langle O \rangle$ predstavlja očekivanu vrijednost promatrane veličine O, a $|\psi\rangle$ je kvantno stanje pripremljeno pomoću klase `QuantumCircuit`. Egzaktan izračun postiže se izvođenjem svih potrebnih linearno algebarskih izračuna kako je implementirano u metodi `Statevector.expectation_value` [21] [22].

2. **Simulacija idealnog kvantnog računala**: Idealno kvantno računalo nema vanjskih smetnji ili grešaka u računanju. Međutim, zbog konačnog broja mjerenja očekivane vrijednosti će fluktuirati oko prave očekivane vrijednosti. `Estimator` to uzima u obzir tako da vrijednosti mjerenja uzorkuje prema normalnoj razdiobi s greškom koja odgovara standardnoj devijaciji takve aproksimirane normalne razdiobe [21].

3. **Simulacija kvantnog računala s šumom**: U ovoj metodi uzimamo u obzir stvarne šumove s pravog kvantnog računala i uključujemo ih u simulaciju.

5.1 Testni model spinskog hamiltonijana

U ovom primjeru, hamiltonijan je već izražen kroz tenzorsku sumu spinskih operatora X, Y i Z. Time izbjegavamo korake potrebne za preslikavanje hamiltonijana iz fermionskog prostora u prostor spinskih operatora.

Razmatrat ćemo VQE metodu na hamiltonijanu $H = 0.7I \otimes X + 0.5I \otimes Y + 0.8X \otimes Z$. Važno je napomenuti da je ovaj hamiltonijan nasumično odabran i nema posebnu fizikalnu interpretaciju. Budući da nema fizikalnu osnovu, konačnoj energiji neće biti dodijeljena specifična jedinica energije.

Promotrimo karakteristike ovog hamiltonijana. Ima tri tenzorska produkta. Prva dva imaju Paulijevu težinu 1, dok zadnji ima težinu 2. Ukupan broj qubita je 2, jer svaki Paulijev operator djeluje na jedan qubit, a u svakom tenzorskom produktu imamo po dva Paulijeva operatora.

Ovaj primjer koristimo kako bismo demonstrirali rad VQE algoritma. Tijekom primjene VQE metode, koristit ćemo različite optimizatore kako bismo usporedili njihove performanse. Slično tome, usporedit ćemo i različite ansatze. Kroz rezultate će biti vidljivo kako različite vrste simulacije utječu na rezultate.

5.1.1 Egzaktna simulacija, usporedba optimizatora

Prođimo kroz kod koji izvršava ovu simulaciju.
Qiskit je moćan alat pa je većina algoritama i optimizatora već pripremljena i dostupna. U ovom radu stoga koristimo već dostupne Qiskit module koji nam omogućuju brzo rješavanje. Svaki kod počinjemo s uvođenjem potrebnih modula [6].

```
from qiskit.quantum_info import SparsePauliOp
from qiskit.primitives import Estimator
from qiskit.algorithms.minimum_eigensolvers import VQE,
                                        NumPyMinimumEigensolver
#ovdje je moguce uvesti i druge optimizatore koji se mogu pronaci na
                                        dokumentaciji qiskita
from qiskit.algorithms.optimizers import COBYLA, L_BFGS_B, SLSQP, SPSA, TNC
from qiskit.circuit.library import TwoLocal
from qiskit.utils import algorithm_globals
from qiskit.algorithms.gradients import FiniteDiffEstimatorGradient
import pylab
```

Nakon toga definiramo hamiltonijan $H = 0.7I \otimes X + 0.5I \otimes Y + 0.8X \otimes Z$ za to koristimo klasu `SparsePauliOp` iz Qiskita. Ova klasa omogućava reprezentaciju hamiltonijana kroz tenzorske produkte Paulijevih operatora, što nam pruža kompaktniji i efikasniji način za rad s kvantnim operatorima.

```
hamiltonijan = SparsePauliOp.from_list(
  [
  ("IX", 0.7),
  ("IY", 0.5),
  ("XZ", 0.8)
  ]
  )
```

Zatim koristimo klasičnu metodu za izračun energije osnovnog stanja. To postižemo korištenjem klase `NumPyMinimumEigensolver` koja je dostupna unutar Qiskita. Ova klasa iskorištava funkcionalnosti `NumPy` modula kako bi putem dijagonalizacije izračunala svojstvene vrijednosti matrice. Na ovaj način dobivamo referentnu vrijednost energije osnovnog stanja, koju kasnije koristimo za usporedbu s rezultatima dobivenim simulacijom.

Važno je napomenuti da, iako je ova metoda korisna za jednostavnije probleme poput onih koje ćemo razmatrati u ovom radu, nije praktična za složenije sustave. U slučaju kompleksnijih problema, matrice koje se dijagonaliziraju postaju prevelike i time računalno zahtjevnije. Međutim, za potrebe ovog rada, ova metoda pruža korisnu referentnu točku za evaluaciju uspješnosti VQE algoritma [23].

```
numpy_solver = NumPyMinimumEigensolver()
result_classical = numpy_solver.compute_minimum_eigenvalue(operator=
                                        hamiltonijan)
ref_value = result_classical.eigenvalue.real
print(f"Referentna vrijednost: {ref_value:.7f}")
```

Potom inicijaliziramo objekt klase `Estimator`, koji predstavlja ključnu komponentu naše simulacije. `Estimator` je klasa unutar Qiskita čija je glavna funkcija aproksimacija očeki-

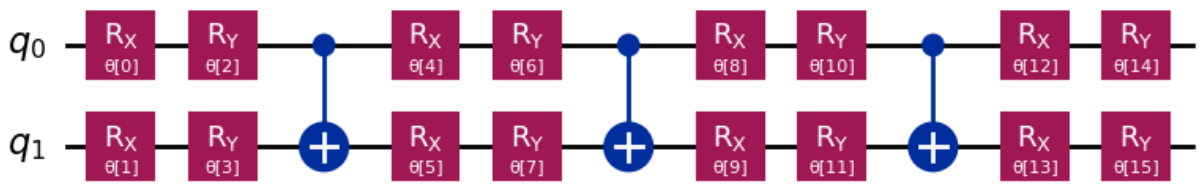
vanih vrijednosti na temelju ulaznih kvantnih krugova i operatora. Ova klasa nudi različite mogućnosti konfiguracije putem svojih argumenata. Ako ne specificiramo nikakve argumente prilikom kreiranja objekta onda radimo na egzaktnom simulatoru [21].

```
estimator = Estimator()
```

Zatim inicijaliziramo skup optimizatora koji će se koristiti za optimizaciju parametara ansatza. Konkretno, koristimo pet različitih optimizatora: `SPSA`, `COBYLA`, `L_BFGS_B`, `SLSQP` i `TNC`. Podsjetimo da `SPSA` i `COBYLA` ne zahtijevaju direktno izračunavanje gradijenta, što može pružiti dodatnu zaštitu od šumova u kvantnim krugovima.

Za konstrukciju ansatza koristimo `TwoLocal` klasu iz Qiskita. Ovo je hardverski efikasan ansatz, temeljen na ponavljajućim blokovima rotacija i sprezanja [24]. U ovom primjeru, ansatz koristi rotacijske operatore R_x i R_y, te CX operator za sprezanje između qubita. Argument `rotation_blocks = ['rx', 'ry']` nam govori da na svaki qubit djelujemo prvo s parametriziranim R_x operatorom pa potom s parametriziranim R_y operatorom. Argumentom `entaglement_blocks` navodimo da potom sprežemo prvi i drugi qubit. S opcijom `reps=3`, postavljamo da se operator sprezanja pojavi tri puta. Kako oko njega uvijek mora biti rotacijski blok to rezultira sa 16 parametara za optimizaciju [24]. Grafički prikaz ansatza može se vidjeti na Slici 13.

```
optimizers = [COBYLA(maxiter=100), L_BFGS_B(maxiter=100), SLSQP(maxiter=100
                                    ), SPSA(maxiter=100), TNC(maxiter=100
                                    )] #ovdje kao argumente postavljamo
                                    maksimalan broj iteracija za pojedini
                                     optimizator
#definiramo liste za spremanje rezultat i medurezultata
converge_counts = []
converge_vals = []
final_values = []
ansatz = TwoLocal(rotation_blocks=['rx', 'ry'], entanglement_blocks='cx',
                                    reps=3)
ansatz.decompose().draw('mpl', style='iqx')
```



Slika 18: *Ansatz konstruiran pomoću `TwoLocal` klase iz Qiskita. Koristi rotacijske operatore R_x i R_y te CX operator za sprezanje. Ukupno ima 16 parametara za optimizaciju.*

Konačno implementiramo VQE algoritam. Kroz petlju, algoritam mijenja optimizatore i bilježi rezultate za svaki od njih. Definiramo funkciju u koju bilježimo međuvrijednosti tijekom optimizacije, što nam omogućuje praćenje iznosa osnovne energije tijekom izvođenja simulacije.

Također inicijaliziramo objekt `gradient` klase `FiniteDiffEstimatorGradient`. Ova klasa omogućava nam da definiramo korak gradijenta, odnosno da postavimo vrijednost ε na neku konačnu vrijednost i preko toga računamo gradijent. Nadalje, kreiramo objekt `vqe` klase `VQE`, gdje kao argumente šaljemo prethodno definirani `estimator`, `ansatz`, funkciju za bilježenje međurezultata i `gradient`. Koristeći metodu `compute_minimum_eigenvalue` unutar `VQE` klase, izračunavamo minimalnu svojstvenu vrijednost hamiltonijana primjenom VQE metode i bilježimo rezultate za svaki optimizator. Na kraju, ispisujemo dobivene vrijednosti.

```
for optimizer in optimizers:
  algorithm_globals.random_seed = 6

  counts = []
  values = []

  def store_intermediate_result(eval_count, parameters, mean, std):
  counts.append(eval_count)
  values.append(mean)

  gradijent = FiniteDiffEstimatorGradient(estimator, epsilon=0.01)

  vqe = VQE(estimator, ansatz, optimizer, callback=
                                        store_intermediate_result, gradient=
                                        gradijent)
  result = vqe.compute_minimum_eigenvalue(operator=hamiltonijan)
  converge_counts.append(counts)
  converge_vals.append(values)
  final_values.append(result.eigenvalue.real)
  print(f"Optimizator: {type(optimizer).__name__}, Konacna energetska
                                        vrijednost: {result.eigenvalue.real:.
                                        7f}")
  print(f"Broj evaluacija za {type(optimizer).__name__}: {len(counts)}")
```

U konačnici kreiramo grafičke prikaze rezultata. Na istom grafu uspoređujemo performanse različitih optimizatora. Na y-osi grafa prikazana je apsolutna razlika između referentne energije i energije dobivene kroz VQE algoritam, dok x-os prikazuje broj evaluacija.

```
pylab.rcParams["font.size"] = 14

# Prvi graf
pylab.figure(figsize=(12, 6))
for i, optimizer in enumerate(optimizers):
pylab.plot(converge_counts[i], abs(ref_value - converge_vals[i]), label=
                                        type(optimizer).__name__)
pylab.xlabel("Broj evaluacija")
```

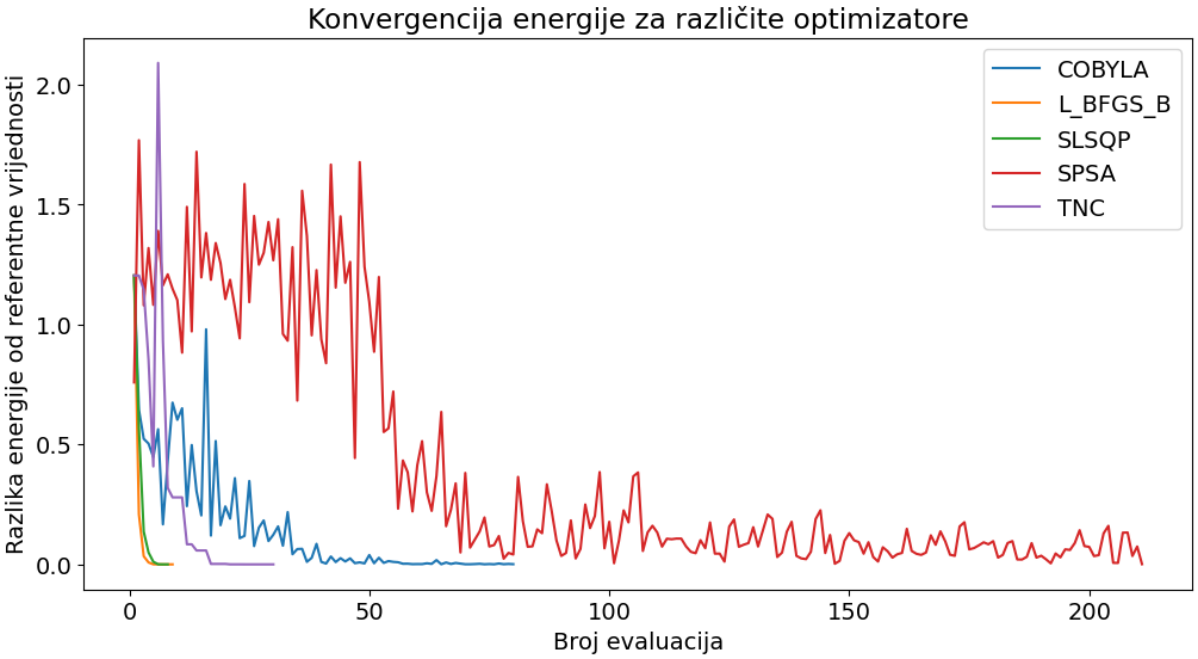
```
pylab.ylabel("Razlika energije od referentne vrijednosti")
pylab.title("Konvergencija energije za razlicite optimizatore")
pylab.legend(loc="upper right")
pylab.show()

# Drugi graf
pylab.figure(figsize=(12, 6))
for i, optimizer in enumerate(optimizers):
pylab.plot(converge_counts[i], abs(ref_value - converge_vals[i]), label=
                                    type(optimizer).__name__)
pylab.ylim(0, 0.01)
pylab.xlim(min(converge_counts[0]), max(converge_counts[0]))
pylab.xlabel("Broj evaluacija")
pylab.ylabel("Razlika energije od referentne vrijednosti")
pylab.title("Konvergencija energije vrlo blizu nule")
pylab.legend(loc="upper right")
pylab.show()
```

Sada ćemo analizirati dobivene rezultate. U tablici 2 možemo primijetiti da optimizatori SPSA

Optimizator	E_{VQE}	n_e	$E_{VQE} - E_{REF}$
COBYLA	-1.1743650	80	0.0003690
L_BFGS_B	-1.1747340	9	0.0000000
SLSQP	-1.1747340	8	0.0000000
SPSA	-1.1738070	211	0.0009270
TNC	-1.1747340	30	0.0000000

Tablica 2: *Tablica prikazuje rezultate različitih optimizatora korištenih u VQE algoritmu za određivanje energije osnovnog stanja hamiltonijana* $H = 0.7I \otimes X + 0.5I \otimes Y + 0.8X \otimes Z$ *preko egzaktne simulacije. Za svaki optimizator prikazana je konačna vrijednost energije* (E_{VQE})*, broj evaluacija* (n_e) *te razlika u od referentne vrijednosti energije* ($E_{VQE} - E_{REF}$)*, gdje referentna vrijednost energije iznos* $E_{REF} = -1.1747340$.

i COBYLA ne postižu najbolje rezultate. Zahtijevaju više evaluacija i ne konvergiraju točno prema referentnoj vrijednosti, iako su njihovi rezultati relativno blizu točne vrijednosti. S druge strane, optimizatori L_BFGS_B, SLSQP i TNC uspješno konvergiraju prema referentnoj vrijednosti, pri čemu SLSQP postiže konvergenciju najbrže. Budući da se radi o egzaktnoj simulaciji, očekivano je da optimizatori koji izračunavaju gradijent, poput L_BFGS_B, SLSQP i TNC, postižu točne rezultate. Za razliku od njih, SPSA i COBYLA ne računaju gradijent direktno, već koriste alternativne metode, što može objasniti manja odstupanja od referentne vrijednosti. Grafički prikaz sa slike 19 jasno potvrđuje što i rezultati iz tablice 2, vidimo da su optimizatori koji ne računaju direktno gradijent mnogo sporiji. Primijetimo još da postoji razlika u broju evaluacija i broju iteracija. To je zato što neki optimizator unutar jedne iteracije mogu više puta evaluirati objektivnu funkciju.



Slika 19: *Grafički prikaz rezultata različitih optimizatora korištenih u VQE algoritmu za određivanje energije osnovnog stanja hamiltonijana* $H = 0.7I \otimes X + 0.5I \otimes Y + 0.8X \otimes Z$ *preko egzaktne simulacije. Na y-osi je prikazana razlika energije od referente vrijednosti, dok je na x-osi prikazan broj evaluacija. Prikazani su* `SPSA`*,* `COBYLA`*,* `L_BFGS_B`*,* `SLSQP` *i* `TNC` *optimizatori, primijetimo da* `COBYLA` *i* `SPSA` *sporije konvergiraju.*

5.1.2 Egzaktna simulacija, usporedba ansatza

Zamjena ansatza tijekom izvršavanja ne rezultira promjenama u dobivenim rezultatima, što je očekivano zbog jednostavnosti početnog hamiltonijana. Da bismo promijenili ansatz, jednostavno trebamo modificirati odgovarajuću liniju koda u našem programu.

```
ansatz = TwoLocal(rotation_blocks=['rx', 'ry'], entanglement_blocks='cx',
                                        reps=3)
```

U Qiskitovoj dokumentaciji navedeni su različiti hardverski efikasni ansatzi koje možemo koristiti [6]. Za ovaj problem, preporučljivo je koristiti upravo takve ansatze s obzirom na to da je naš hamiltonijan već izražen pomoću Paulijevih operatora.

5.1.3 Simulacija na idealnom kvantom računalu, usporedba optimizatora

Kod za izvršavanje simulacije na idealnom kvantnom računalu vrlo je sličan prethodno navedenom kodu. Glavna razlika je u sljedećoj liniji koda.

```
estimator = Estimator(options={"shots": 10000})
```

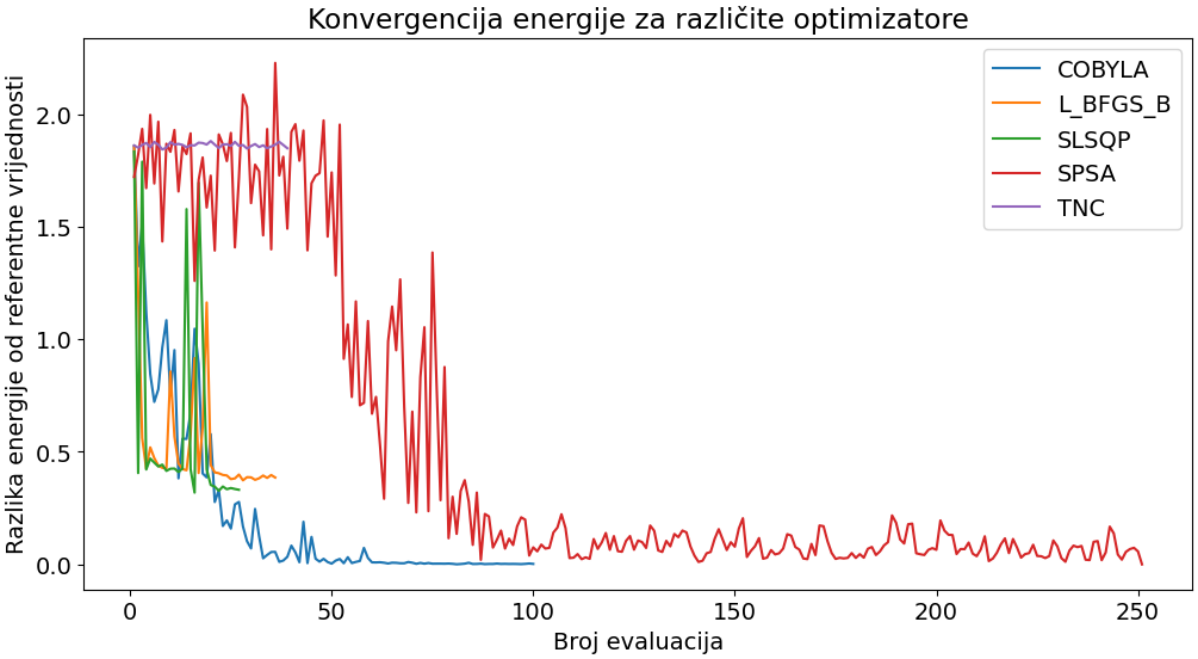
Ovdje, kao argument za `estimator`, postavljamo da želimo simulaciju na idealnom kvantnom računalu gdje aproksimaciju očekivane vrijednosti radimo s 10000 mjerenja. Ovo znači da za

svaku evaluaciju objektivne funkcije koju napravi određeni optimizator idealna simulacija mora napraviti 10000 mjerenja. Primjerice ako neki optimizator napravi 1000 evaluacija to znači ukupno radimo 10 milijuna mjerenja. U kvantnoj kemiji, definira se kemijska preciznost kao odstupanje od 4.2kJ/mol. To znači da dobivena energija molekule može odstupati od stvarne energije za najviše 4.2kJ/mol kako bi bila korisna u praktične svrhe [25].
Na pravim kvantnim računalima kako bi se postigla ova točnost, primjerice za molekulu CO_2 potrebno je obaviti čak 32 milijarde mjerenja za samo jednu aproksimaciju očekivane energije. Ovo uzima u obzir i korištenje metoda koje smanjuju ukupan broj potrebnih mjerenja. Dakle, kvantno računalo s kapacitetom od 128 qubita trebalo bi neprekidno raditi 39 dana za jednu takvu aproksimaciju. S obzirom na to da najbolji optimizatori danas zahtijevaju barem nekoliko stotina evaluacija, dolazimo do zaključka o ogromnom broju potrebnih mjerenja, ovo je jedna od velikih prepreka VQE algoritmu [26].
U ovom radu neće se koristiti metode za smanjenje broja mjerenja budući da se fokusiramo na osnovne primjere. Međutim na ovome hamiltonijanu, demonstrirat ćemo kakve rezultate daju optimizatori kada im ograničimo broj mjerenja na 300 za jednu aproksimaciju.

Optimizator	**10000 mjerenja**			**300 mjerenja**		
	E_{VQE}	n_e	$E_{VQE} - E_{REF}$	E_{VQE}	n_e	$E_{VQE} - E_{REF}$
COBYLA	-1.1742307	100	0.0005033	-1.1292275	100	0.0455065
L_BFGS_B	-0.7889010	36	0.3858330	0.7397136	11	1.9144476
SLSQP	-0.8432546	27	0.3314794	-0.3620885	200	0.8126455
SPSA	-1.1746451	251	0.0000889	-1.1729421	251	0.0017919
TNC	0.6738757	39	1.8486097	0.6137517	23	1.7884857

Tablica 3: *Tablica prikazuje rezultate različitih optimizatora korištenih u VQE algoritmu za određivanje osnovnog stanja hamiltonijana $H = 0.7I \otimes X + 0.5I \otimes Y + 0.8X \otimes Z$ preko simulacije na idealnom kvantom računalu. Za svaki optimizator prikazana je konačna vrijednost energije (E_{VQE}), broj evaluacija (n_e), te razlika u odnosu na referentnu vrijednost energije ($E_{VQE} - E_{REF}$), gdje referentna vrijednost energija iznosi $E_{REF} = -1.1747340$. Analiza je napravljena za 10000 mjerenja i za 300 mjerenja u svakom izračunu.*

U tablici 3 prikazani su rezultati. Vidimo prvo da simulacija sa 10000 i sa 300 mjerenja u svakom izračunu daju rezultate značajno različite u usporedbi s onima dobivenim za egzaktnu simulaciju. Ova razlika može proizaći iz više razloga. Budući da sada koristimo mjerenja za aproksimaciju vrijednosti, optimizatori `L_BFGS_B`, `SLSQP` i `TNC` mogu lako zapasti u lokalne minimume. Drugim riječima, ovi optimizatori su osjetljiviji na problem "neplodne visoravni", dok `SPSA` i `COBYLA` imaju veću vjerojatnost da izbjegnu lokalne minimume jer ne računaju gradijent izravno.
Primijetimo i kako `SPSA` najotpornija na smanjenje broja mjerenja. Postoji i mogućnost da su inicijalne vrijednosti uzrok lošoj izvedbi nekih optimizatora. U ovom konkretnom primjeru, inicijalne vrijednosti ne postavljamo eksplicitno, već ih biramo nasumično. Sljedeći linija koda postavlja inicijalne vrijednosti temeljene na određenom sjemenu (engl. "seed").



Slika 20: *Grafički prikaz rezultata različitih optimizatora korištenih u VQE algoritmu za određivanje energije osnovnog stanja hamiltonijana $H = 0.7I \otimes X + 0.5I \otimes Y + 0.8X \otimes Z$ preko simulacije na idealnom kvantom računalu. Na y-osi je prikazana razlika energije od referente vrijednosti, dok je na x-osi prikazan broj evaluacija. Prikazani su* `SPSA`*,* `COBYLA`*,* `L_BFGS_B`*,* `SLSQP` *i* `TNC` *optimizatori, primijetimo kako ovdje najbolje rezultate daju* `SPSA` *i* `COBYLA` *za razliku od egzaktne simulacije. Analiza za 10000 mjerenja.*

```
algorithm_globals.random_seed = 6
```

Možemo analizirat izvedbu različitih optimizatori s obzirom na različite inicijalne vrijednosti. Da bismo to postigli, možemo prilagoditi kod tako da cijeli eksperiment ponovimo 100 puta, svaki put koristeći različito sjeme za generiranje inicijalnih vrijednosti. Ova analiza može nam pružiti uvid u stabilnost i pouzdanost svakog optimizatora u različitim scenarijima. Implementacija ovog eksperimenta može se postići postavljanjem cijelog koda unutar petlje koja se izvršava 100 puta, gdje se pri svakom prolasku kroz petlju nasumično odabire sjeme. U konačnici računamo srednju vrijednosti energije osnovnog stanja za svaki optimizator i pripadnu standardnu devijaciju. Ovo radimo za simulaciju sa 300 i 10000 mjerenja. Iz tablice 4 možemo uočiti da optimizatori `COBYLA` i `SPSA` konzistentno daju rezultate koji su blizu referentne vrijednosti za 10000 mjerenja, i to s niskim standardnim devijacijama. To sugerira da su ovi optimizatori robustni i pouzdani, bez obzira na početne vrijednosti. S druge strane, ostali optimizatori pokazuju značajno veće standardne devijacije, što ukazuje na njihovu varijabilnost u rezultatima. Ovi optimizatori nisu najpogodniji za ovu vrstu problema, s obzirom na njihovu nepouzdanost u različitim scenarijima. Ponovno vidimo da smanjenjem mjerenja na 300 svi optimizatori gube na točnosti, no `SPSA` daje i dalje rezultate gdje je referentna vrijednost unutar jedne standardne devijacije i gdje je standardna devijacija mala u odnosu na druge optimizatore.

Optimizator	10000 mjerenja		300 mjerenja	
	E_{AVG}	$E_{AVG} - E_{REF}$	E_{AVG}	$E_{AVG} - E_{REF}$
COBYLA	-1.174 ± 0.003	0.00051	-1.160 ± 0.057	0.01458
L_BFGS_B	-0.769 ± 0.433	0.40559	-0.152 ± 0.611	1.02304
SLSQP	-0.975 ± 0.259	0.19964	-0.565 ± 0.477	0.61005
SPSA	-1.174 ± 0.001	0.00091	-1.172 ± 0.006	0.00231
TNC	-0.054 ± 0.529	1.12119	0.646 ± 0.082	0.52890

Tablica 4: *Prikaz performansi različitih optimizatora temeljenim na 100 ponavljanja cijele simulacije s različitim inicijalnim vrijednostima. Analiza uključuje srednju vrijednost energije osnovnog stanja i standardnu devijaciju (E_{AVG}) za svaki optimizator. Energije su dobivene VQE metodom za hamiltonijan $H = 0.7I \otimes X + 0.5I \otimes Y + 0.8X \otimes Z$ preko simulacije na idealnom kvantom računalu. Prikazana je i razlika prosječne energije od referentne ($E_{AVG} - E_{REF}$), gdje je $E_{REF} = -1.1747340$. Analiza je napravljena za 10000 mjerenja i za 300 mjerenja u svakom izračunu.*

5.1.4 Simulacija sa šumovima

Razmotrimo sada simulaciju s uključenim šumovima, modeliranu prema stvarnom kvantnom računalu IBM-a. Ovu simulaciju implementirat ćemo na dva različita načina: prvo lokalno izvršavanje, a zatim izvršavanje putem IBM-ovih klasičnih računala preko oblaka. Odlučili smo se za ovakav pristup jer implementacija putem IBM-ovog `qiskitRuntimeService` nudi mogućnost mitigacije grešaka. Time možemo analizirati utjecaj ove mogućnosti na konačne rezultate. U ovim simulacijama koristit ćemo isključivo `SPSA` optimizator, budući da se u prethodnim simulacijama na idealnom kvantnom računalu pokazao kao najrobustniji. Za ansatz ćemo ponovno koristiti `TwoLocal` klasu.

Sada koristimo nekoliko novih modula. Iako ponovno koristimo `Estimator`, sada ga uvozimo iz `qiskit_aer.primitives` umjesto iz `qiskit.primitives` . Ključna razlika leži u tome što se sada oslanjamo na Qiskit Aer, koji nudi kvantne simulatore s realističnim opcijama za modeliranje šumova. Dodatno, iz Aer paketa uvozimo `NoiseModel` klasu koju koristimo za definiranje šumova [27]. Konačno, koristimo i `FakeVigo` klasu koja simulira šumove karakteristične za stvarno kvantno računalo IBM-a pod nazivom Vigo [27]. Radi se o kvantnom računalu s 5 qubita[28] .

```
from qiskit.quantum_info import SparsePauliOp
from qiskit.algorithms.minimum_eigensolvers import VQE,
                                        NumPyMinimumEigensolver
from qiskit.algorithms.optimizers import SPSA
from qiskit.circuit.library import TwoLocal
from qiskit.utils import algorithm_globals
from qiskit.algorithms.gradients import FiniteDiffEstimatorGradient
from qiskit_aer.primitives import Estimator as AerEstimator
from qiskit_aer.noise import NoiseModel
from qiskit.providers.fake_provider import FakeVigo #ovdje se moze odabrati
                                         bilo koje drugo kvanto racunalo IBM-
                                        a
```

```
import pylab
```

Zatim kreiramo objekt klase `FakeVigo()`, koji koristimo za definiranje modela šuma. Varijabla `coupling_map` pohranjuje informacije o spregnutosti kvantnog računala. Ova informacija je ključna jer nam je potrebno prilagoditi kvantni krug specifičnom hardveru odabranog kvantnog računala. Budući da u ovom kontekstu radimo s kvantnim računalom Vigo, preuzimamo njegovu hardversku konfiguraciju i prilagođavamo kvantni krug za izvršavanje na njemu, odnosno radimo transpilaciju kvantnog kruga.

Nadalje, u varijabli `noise_model` pohranjujemo profil šuma kvantnog računala Vigo. Nakon toga, kreiramo objekt `noisy_estimator`, kojem prosljeđujemo prethodno definirane varijable, čime prilagođavamo `noisy_estimator` za rad s specifičnim šumovima kvantnog računala Vigo. Važno je napomenuti da koristimo metodu `density_matrix`, što znači da problem evaluiramo koristeći matrice gustoće. Ovo je posebno korisno kada radimo sa šumovima, jer često dolazimo do mješovitih stanja, a evaluacija postaje jednostavnija koristeći matrice gustoće. Dodatno, postavljamo da se mjerenje izvršava 1000 puta. Broj mjerenja postavljen je relativno nisko kako bismo kasnije mogli usporediti rezultate s simulacijom u kojoj se provodi korekcija grešaka. Budući da je ta simulacija računalno zahtjevnija, broj mjerenja je postavljen na ovako nisku vrijednost kako bi se simulacija izvršila u razumnom vremenskom okviru.

```
device = FakeVigo()
coupling_map = device.configuration().coupling_map
noise_model = NoiseModel.from_backend(device)

noisy_estimator = AerEstimator(
backend_options={
  "method": "density_matrix",
  "coupling_map": coupling_map,
  "noise_model": noise_model
  },
  run_options={"shots": 1000},
  transpile_options={"seed_transpiler": seed},
  )
```

Slijedi još kod koji je vrlo sličan onome koji smo prethodno razmatrali. Ključna razlika je u tome što `vqe` objektu kao argument prosljeđujemo `noisy_estimator` umjesto `estimator`.

```
hamiltonijan = SparsePauliOp.from_list(
  [
  ("IX", 0.7),
  ("IY", 0.5),
  ("XZ", 0.8)
  ]
```

```
  )

# Classical reference value
numpy_solver = NumPyMinimumEigensolver()
result_classical = numpy_solver.compute_minimum_eigenvalue(operator=
                                        hamiltonijan)
ref_value = result_classical.eigenvalue.real
print(f"Referentna vrijednost: {ref_value:.7f}")

optimizer = SPSA(maxiter=1000)
counts = []
values = []

def store_intermediate_result(eval_count, parameters, mean, std):
  counts.append(eval_count)
  values.append(mean)

ansatz = TwoLocal(rotation_blocks=['rx', 'ry'], entanglement_blocks='cx',
                                        entanglement='full', reps=1)
gradient = FiniteDiffEstimatorGradient(noisy_estimator, epsilon=0.01)

# VQE
vqe = VQE(noisy_estimator, ansatz, optimizer, callback=
                                        store_intermediate_result, gradient=
                                        gradient)
result = vqe.compute_minimum_eigenvalue(operator=hamiltonijan)
print(f"Optimizator: {type(optimizer).__name__}, Konacna vrijednost
                                        energije: {result.eigenvalue.real:.7f
                                        }")
print(f"Broj evaluacija za {type(optimizer).__name__}: {len(counts)}")
```

Optimizator	E_{VQE}	n_e	$E_{VQE} - E_{REF}$
SPSA	-0.9088000	2051	0.2659340

Tablica 5: *Prikaz energije (E_{VQE}) SPSA optimizatora dobivene VQE metodom za hamiltonijan $H = 0.7I \otimes X + 0.5I \otimes Y + 0.8X \otimes Z$ za simulaciju sa šumovima, gdje koristimo 1000 mjerenja. Primijetimo kako šumovi značajno utječu na konačnu vrijednost energije osnovnog stanja u odnosu na idealnu simulaciju. Referenta energija iznosi $E_{REF} = -1.1747340$*

U tablici 5 vidimo rezultate ovog izvršavanja, vidljivo je da šumovi znatno utječu na konačan rezultat. Odstupanje sa šumovima uz 1000 iteracija znatno je veće od odstupanja idealne simulacije sa samo 300 mjerenja (tablica 4).

5.1.5 Implementacija s mitigacijom grešaka

U ovom dijelu razmatrat ćemo implementaciju koja uključuje mitigaciju grešaka. Ova funkcionalnost postiže se korištenjem ugrađenih opcija Qiskit-a. Važno je napomenuti da su ove opcije dostupne samo kada se koristi IBM-ov `QiskitRuntimeService`, odnosno kada se spojimo preko oblaka na klasična računala IBM-a koja potom izvršavaju simulaciju [29]. U nastavku će biti predstavljeni samo dijelovi koda koji se razlikuju od prethodno opisanog primjera.
Umjesto korištenja `Aer` simulatora i `estimatora`, koristimo `estimator` iz `qiskit_ibmruntime`. Uvozimo nekoliko ključnih klasa: `QiskitRuntimeService`, `Session` i `Options`. Klasa `QiskitRuntimeService` omogućuje nam povezivanje s IBM-ovim računalima. `Session` koristimo kako bismo osigurali uspješno izvršavanje na IBM-ovim računalima putem oblaka, dok `Options` koristimo za postavljanje različitih opcija simulacije [29].

```
from qiskit.quantum_info import SparsePauliOp
from qiskit.algorithms.minimum_eigensolvers import VQE,
                                        NumPyMinimumEigensolver
from qiskit.algorithms.optimizers import SPSA
from qiskit.circuit.library import TwoLocal
from qiskit.utils import algorithm_globals
from qiskit.algorithms.gradients import FiniteDiffEstimatorGradient
from qiskit_ibm_runtime import QiskitRuntimeService, Estimator, Session,
                                        Options #ovdje kljucna razlika u
                                        odnosu na prosli primjer
from qiskit_aer.noise import NoiseModel
from qiskit.providers.fake_provider import FakeVigo
```

Zatim je postupak vrlo sličan kao za simulaciju bez mitigacije grešaka, s ključnom razlikom u konfiguraciji simulacije. Kao i ranije, postavljamo model šuma te konfiguraciju hardvera kvantnog računala. Ponovno koristimo kvantno računalo Vigo. Međutim, sada koristimo dodatnu opciju unutar objekta `options` pod nazivom `resiliance_level`. Ova opcija omogućava postavljanje razine otpornosti na greške tijekom simulacije. Više razine otpornosti imaju za cilj postizanje točnijih rezultata, ali uz cijenu dužeg vremena izvođenja. Postoje četiri dostupne razine otpornosti, pri čemu je 3 najviša razina. U ovom primjeru koristili smo razinu 2 koja koristi metodu "Zero Noise Extrapolation (ZNE)". Ova metoda ima za cilj smanjiti pristranost, iako nije zajamčeno da će rezultati biti potpuno nepristrani [30]. Dalje, koristeći objekt `service`, povezujemo se s IBM Quantum Experience, internetskom platformom koja pruža javni pristup kvantnim računalnim uslugama. Za pozadinu (engl. "backend") odabiremo `ibmq_qasm_simulator`, koji je snažan simulator dostupan putem navedene usluge. Broj mjerenja postavljamo na 1000 jer je ova simulacija računalno zahtjevna.

```
fake_backend = FakeVigo()
noise = NoiseModel.from_backend(fake_backend)
```

```
options = Options()
options.simulator = {
    "noise_model": noise_model,
    "basis_gates": fake_backend.configuration().basis_gates,
    "coupling_map": fake_backend.configuration().coupling_map,
    "seed_simulator": 42
    }

options.execution.shots = 1000
options.resilience_level = 2

service = QiskitRuntimeService(channel="ibm_quantum")
backend = service.backend("ibmq_qasm_simulator")
```

Ostatak koda je jednak jedino što izvršavamo algoritam preko `Session` klase.

```
hamiltonijan = SparsePauliOp.from_list(
  [
  ("IX", 0.7),
  ("IY", 0.5),
  ("XZ", 0.8)
  ]
  )
numpy_solver = NumPyMinimumEigensolver()
result_classical = numpy_solver.compute_minimum_eigenvalue(operator=
                                        hamiltonijan)
ref_value = result_classical.eigenvalue.real
print(f"Referentna vrijednost: {ref_value:.7f}")

#kada idemo preko runtime-a potrebno je kroz klasu Session izvrsavati
                                        program
with Session(service=service, backend=backend) as session:
  estimator = Estimator(session=session,options=options)

  #ostatak jednak

  session.close() #potrebno zatvoriti session
```

Optimizator	E_{VQE}	n_e	$E_{VQE} - E_{REF}$
SPSA	-1.1356167	2051	0.0391173

Tablica 6: *Prikaz rezultata `SPSA` optimizatora dobivene VQE metodom za hamiltonijan* $H = 0.7I \otimes X + 0.5I \otimes Y + 0.8X \otimes Z$ *za simulaciju sa šumovima gdje imamo uključenu mitigaciju greške. Primijetimo usporedbom iz tablice 4 kako mitigacija greške značajno poboljša rezultat. Referenta energija iznosi* $E_{REF} = -1.1747340$.

Iz tablice 6 vidimo da se rezultat izvršavanja znatno poboljšao u odnosu na simulaciju bez mitigacije grešaka. Kada koristimo mitigaciju grešaka, VQE algoritam postiže 96.7% točnosti u odnosu na referentnu vrijednost. Dok bez mitigacije grešaka postižemo 77.4% točnosti.

5.2 Implementacija za H_2 molekulu

Ovdje primjenjujemo VQE algoritam na molekulu vodika. Razmotrit ćemo dvije različite simulacije: egzaktnu simulaciju i simulaciju na idealnom kvantnom računalu. Budući da simulacija s uključenim šumovima zahtijeva znatno više vremena, nećemo je razmatrati u ovom kontekstu.

Za razliku od prethodnog primjera, sada moramo uzeti u obzir korak preslikavanja iz fermionskog prostora u prostor spinskih operatora. Ovaj korak je ključan jer kvantna računala koriste spinske operatore za izračune.

Dodatno, osim Qiskita, koristit ćemo i PySCF, paket specijaliziran za kvantnu kemiju. PySCF nam omogućuje pristup elektronskoj strukturi molekula. U ovom kontekstu, koristimo PySCF kako bismo dobili hamiltonijan preko fermionskih operatora za molekulu vodika koju promatramo [31]. Napomenimo da PySCF koristi jedinicu energije hartree, koja iznosi

$$1\ \text{Ha}\ \approx 27.2114\ \text{eV}\ \approx 4.35974 \times 10^{-18}\ \text{J}\ . \tag{5.1}$$

Stoga će sve energije dobivene VQE metodom biti prikazane u Hartree jedinicama. Budući da sada radimo s problematikom kvantne kemije za evaluaciju ispravnosti dobivenih energija, koristit ćemo kemijsku preciznost koja u hartree jedinici iznosi 1.6mHa [25].

5.2.1 Egzaktna simulacija, usporedba optimizatora

Usporedit ćemo performanse različitih optimizatora. Odabrano je pet optimizatora, istih kao u prethodnom primjeru. Za ansatz, odlučili smo se za `UCCSD` ansatz, koji se često koristi u kvantnoj kemiji i pokazao se kao jedan od najučinkovitijih za ovu vrstu problema [3].
Trebamo uvesti dodatne module koji nam omogućavaju konfiguraciju molekule H_2. Kao što je već istaknuto, koristimo `UCCSD` ansatz. Dodatno, primjenjujemo klasu `HartreeFock` koja nam omogućava inicijalizaciju početnih vrijednosti ansatza kroz Hartree-Fock metodu. Također, uključujemo klasu `PySCFDriver` koja nam pruža mogućnost dobivanja elektronske strukture molekule i uključujemo klasu `JordaWignerMapper` koju koristimo za preslikavanje.

```
from qiskit_nature.units import DistanceUnit
from qiskit_nature.second_q.drivers import PySCFDriver
from qiskit_nature.second_q.mappers import JordanWignerMapper #ovdje se
                                        moze korsiti i neko drugo
                                        preslikavanje, primjerice
```

```
                                        ParityMapper
from qiskit_nature.second_q.circuit.library import HartreeFock, UCCSD #Za
                                        konfiguraciju i postavljanje
                                        inicijalnih vrijednosti
from qiskit.quantum_info import SparsePauliOp
from qiskit.primitives import Estimator
from qiskit.algorithms.minimum_eigensolvers import VQE,
                                        NumPyMinimumEigensolver
from qiskit.algorithms.optimizers import COBYLA, L_BFGS_B, SLSQP, SPSA, TNC
from qiskit.utils import algorithm_globals
from qiskit.algorithms.gradients import FiniteDiffEstimatorGradient
import pylab
```

Zatim definiramo konfiguraciju naše molekule. Udaljenost među atomima vodika u molekuli H_2 postavljamo na 0.735 Å [32]. Objekt `es_problem` sadrži sve potrebne informacije o konfiguraciji molekule.

```
# Definicija drivera za molekulu
driver = PySCFDriver(
  atom="H 0 0 0; H 0 0 0.735", #koordinate u prostoru
  basis="sto3g",
  charge=0,
  spin=0,
  unit=DistanceUnit.ANGSTROM,
  )

# Pokretanje drivera
es_problem = driver.run()
```

Potom koristimo Jordan-Wignerovo preslikavanje, kako bi hamiltonijan iz fermionskog prostora prebacili u prostor qubita.

```
  mapper = JordanWignerMapper()
  fermionic_op = es_problem.hamiltonian.second_q_op()
  qubit_op = mapper.map(fermionic_op)
```

Kao rezultate se dobije sljedeći hamiltonijan

$$\begin{aligned}\hat{H} = &-0.8105479805373275 \cdot \hat{I} \otimes \hat{I} \otimes \hat{I} \otimes \hat{I} \\ &+0.1721839326191556 \cdot \hat{I} \otimes \hat{I} \otimes \hat{I} \otimes \hat{Z} \\ &-0.2257534922240239 \cdot \hat{I} \otimes \hat{I} \otimes \hat{Z} \otimes \hat{I} \\ &+0.17218393261915554 \cdot \hat{I} \otimes \hat{Z} \otimes \hat{I} \otimes \hat{I} \\ &-0.2257534922240239 \cdot \hat{Z} \otimes \hat{I} \otimes \hat{I} \otimes \hat{I} \\ &+0.12091263261776629 \cdot \hat{I} \otimes \hat{I} \otimes \hat{Z} \otimes \hat{Z} \\ &+0.16892753870087907 \cdot \hat{I} \otimes \hat{Z} \otimes \hat{I} \otimes \hat{Z} \\ &+0.04523279994605784 \cdot \hat{Y} \otimes \hat{Y} \otimes \hat{Y} \otimes \hat{Y} \\ &+0.04523279994605784 \cdot \hat{X} \otimes \hat{X} \otimes \hat{Y} \otimes \hat{Y} \\ &+0.04523279994605784 \cdot \hat{Y} \otimes \hat{Y} \otimes \hat{X} \otimes \hat{X} \\ &+0.04523279994605784 \cdot \hat{X} \otimes \hat{X} \otimes \hat{X} \otimes \hat{X} \\ &+0.16614543256382414 \cdot \hat{Z} \otimes \hat{I} \otimes \hat{I} \otimes \hat{Z} \\ &+0.16614543256382414 \cdot \hat{I} \otimes \hat{Z} \otimes \hat{Z} \otimes \hat{I} \\ &+0.17464343068300445 \cdot \hat{Z} \otimes \hat{I} \otimes \hat{Z} \otimes \hat{I} \\ &+0.12091263261776629 \cdot \hat{Z} \otimes \hat{Z} \otimes \hat{I} \otimes \hat{I}\end{aligned} \tag{5.2}$$

U analizi hamiltonijana za molekulu vodika, uočavamo da je potrebno četiri qubita za njegovu reprezentaciju. Hamiltonijan je predstavljen sumom od petnaest tenzorskih produkata. U usporedbi s prethodno razmatranim hamiltonijanima, ovaj je znatno kompleksniji.
Računamo referentu energiju osnovnog stanja klasičnom metodom. Važno je napomenuti da, dok u VQE-u računamo isključivo elektronski doprinos energiji, za izračun ukupne energije molekule moramo uzeti u obzir i energiju odbijanja jezgri. Konkretno, uzimamo u obzir doprinos energije odbijanja jezgri koji dobivamo iz `es_problem.nuclear_repulsion_energy`.

```
numpy_solver = NumPyMinimumEigensolver()
result_classical = numpy_solver.compute_minimum_eigenvalue(operator=
                                        qubit_op)
ref_value = result_classical.eigenvalue.real + es_problem.
                                        nuclear_repulsion_energy
print(f"Referentna vrijednost: {ref_value:.7f}")
print(f"Doprinos jezgre: {es_problem.nuclear_repulsion_energy:.7f}")
print(f"Elektronski doprinos: {result_classical.eigenvalue.real:.7f}")
```

Zatim konstruiramo `UCCSD` ansatz. Kao ulazne argumente, ansatzu pružamo informacije o konfiguraciji molekule, uključujući broj prostornih orbitala i broj čestica. Ove informacije dobivamo iz `es_problem` objekta. Budući da `UCCSD` ansatz prikazuje rezultat kroz fermionske operatore, također pružamo objekt za preslikavanje, u ovom slučaju koristimo Jordan-Wigner

preslikavanje. Dodatno, kao argument šaljemo inicijalno stanje koje dobivamo primjenom HartreeFock metode. Ovo je korisno jer omogućava VQE algoritmu da traži rješenje blizu vrijednosti dobivenih HartreeFock metodom.

```
estimator = Estimator()
optimizers = [COBYLA(maxiter=80), L_BFGS_B(maxiter=60), SLSQP(maxiter=60),
                                        SPSA(maxiter=80), TNC(maxiter=60)]
converge_counts = []
converge_vals = []
final_values = []
ansatz = UCCSD(
  es_problem.num_spatial_orbitals,
  es_problem.num_particles,
  mapper,
  initial_state=HartreeFock(
    es_problem.num_spatial_orbitals,
    es_problem.num_particles,
    mapper,
    ),
  )
```

Konačno prolazimo kroz sve optimizatore i izvršavamo VQE algoritam, postupak je isti kao i u prošlom primjeru.

```
# Izracun VQE za svaki optimizator
for optimizer in optimizers:
  algorithm_globals.random_seed = 6

  counts = []
  values = []

  # Funkcija za pohranu medurezultata
  def store_intermediate_result(eval_count, parameters, mean, std):
  counts.append(eval_count)
  values.append(mean)

  gradient = FiniteDiffEstimatorGradient(estimator, epsilon=0.01)

  vqe = VQE(estimator, ansatz, optimizer, callback=
                                        store_intermediate_result, gradient=
                                        gradient)
  vqe.initial_point = np.zeros(ansatz.num_parameters) #ovdje postavljamo da
                                         vqe koristi incijalne vrijednosti
                                        dobivene HartreeFock metodom
  result = vqe.compute_minimum_eigenvalue(operator=qubit_op)
  converge_counts.append(counts)
  converge_vals.append(values + es_problem.nuclear_repulsion_energy)
```

```
final_values.append(result.eigenvalue.real + es_problem.
                                    nuclear_repulsion_energy)
print(f"Optimizator: {type(optimizer).__name__}, Konacna vrijednost
                                    energije: {result.eigenvalue.real +
                                    es_problem.nuclear_repulsion_energy:.
                                    7f}")
print(f"Broj evaluacija za {type(optimizer).__name__}: {len(counts)}")
```

Pogledajmo sada rezultate. Iz tablice 7, slično kao u prošlom primjeru vidimo da svi optimiza-

Optimizator	E_{VQE}[Ha]	n_e	$E_{VQE} - E_{REF}$[Ha]
COBYLA	-1.1373060	49	**0.0000000**
L_BFGS_B	-1.1373060	5	**0.0000000**
SLSQP	-1.1373060	4	**0.0000000**
SPSA	-1.1367500	211	**0.0005560**
TNC	-1.1373060	5	**0.0000000**

Tablica 7: *Tablica prikazuje rezultate različitih optimizatora korištenih u VQE algoritmu za određivanje energije osnovnog stanja molekule H_2 preko egzaktne simulacije. Za svaki optimizator prikazana je konačna vrijednost energije (E_{VQE}), broj evaluacija (n_e) te razlika u odnosu na referentnu vrijednost energije osnovnog stanja ($E_{VQE} - E_{REF}$), gdje je $E_{REF} = 1.1373060$Ha referenta energija. U stupcu $E_{VQE} - E_{REF}$ su podebljani rezultati gdje je postignuta kemijska preciznost. Kako se radi o egzaktnoj simulaciji to je svugdje postignuto.*

tori daju odlične rezultate s obzirom da se radi o egzaktnoj simulaciji. Iz slike 16 vidimo da

Slika 21: *Grafički prikaz rezultata različitih optimizatora korištenih u VQE algoritmu za određivanje energije osnovnog stanja molekule H_2 preko egzaktne simulacije. Na y-osi je prikazana razlika energije od referente vrijednosti, dok je na x-osi prikazan broj evaluacija. Prikazani su `SPSA`, `COBYLA`, `L_BFGS_B`, `SLSQP` i `TNC` optimizatori. Primijetimo kako ovdje `SPSA` optimizator radi nasumičnosti oko 50-te evaluacije prestane konvergirati nakon čega ponovno konvergira prema referentnoj vrijednosti.*

vrlo brzo grafovi konvergiraju prema točno vrijednosti, ovo se događa jer smo koristili inicijalne

vrijednosti dobivene HartreeFock metodom.

5.2.2 Egzaktna simulacija, ovisnost energije o udaljenosti među atomima vodika

Razmotrimo sada simulaciju u kojoj variramo udaljenost između atoma vodika i promatramo kako se mijenja energija osnovnog stanja. Ovo postižemo tako da, unutar petlje, mijenjamo udaljenost između atoma vodika koristeći `PySCFDriver`. Nakon svake promjene udaljenosti, izvršavamo VQE algoritam i bilježimo rezultirajuću energiju osnovnog stanja.
Kao i uvijek krećemo s uvozom modula, moduli koje koristimo su slični kao u prethodnom primjeru s iznimkom da koristimo klasu `GroundStateSolver`. Ova klasa pruža kompaktniji kod za implementaciju VQE algoritma. Odlučili smo se za nju jer nas ne zanimaju međurezultati, već isključivo konačna energija osnovnog stanja [33].

```
from qiskit_nature.units import DistanceUnit
from qiskit_nature.second_q.drivers import PySCFDriver
from qiskit_nature.second_q.mappers import ParityMapper
from qiskit_nature.second_q.circuit.library import HartreeFock, UCCSD
from qiskit_nature.second_q.algorithms import GroundStateEigensolver
from qiskit.primitives import Estimator
from qiskit.algorithms.minimum_eigensolvers import VQE,
                                        NumPyMinimumEigensolver
from qiskit.algorithms.optimizers import SPSA
from qiskit.utils import algorithm_globals
import numpy as np
import pylab
```

Potom definiramo petlju koja prolazi kroz različite udaljenosti između atoma vodika. Ispitujemo energiju osnovnog stanja u rasponu od $0.2\,\text{Å}$ do $2.2\,\text{Å}$ s korakom od $0.1\,\text{Å}$. Za preslikavanje koristimo paritetno preslikavanje, kako se radi o jednostavnoj molekuli jednaki rezultati su dobiveni i za Jordan-Wignerovo preslikavanje. Kao ansatz koristimo `UCCSD`, dok za optimizaciju koristimo `SLSQP` optimizator. S obzirom na rezultate iz prethodnog primjera, gdje su svi optimizatori u egzaktnoj simulaciji dali odlične rezultate, mogli smo odabrati bilo koji optimizator. Simulaciju izvršavamo koristeći klasu `GroundStateSolver` zbog njene jednostavnosti.

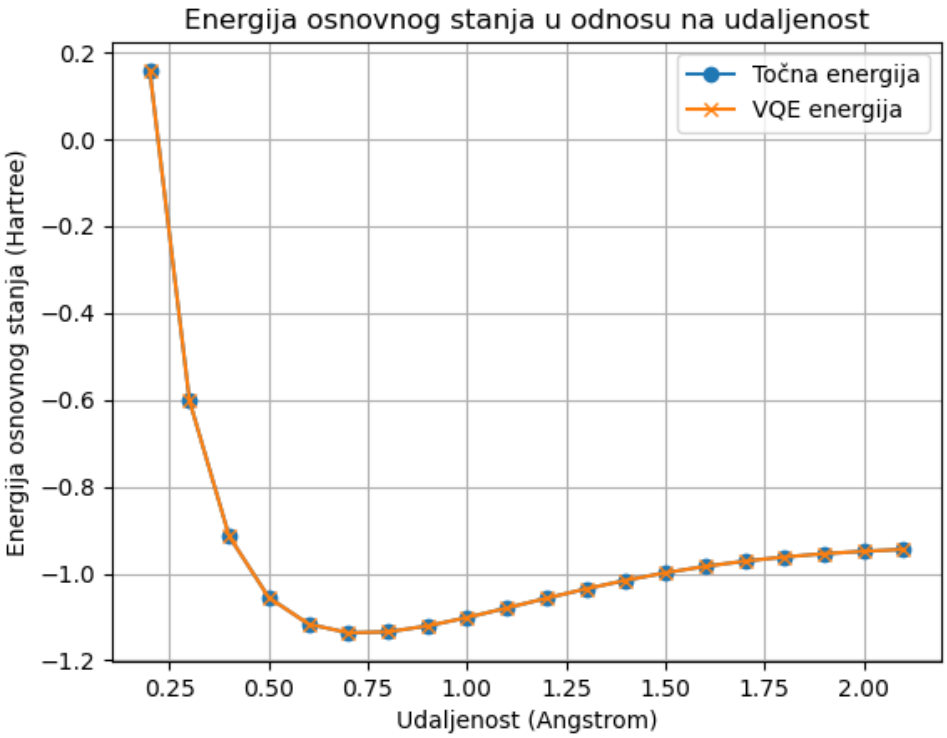
```
for distance in distances: #petlja u kojoj mijenjamo udaljenost
  driver = PySCFDriver(
    atom=f"H 0 0 0; H 0 0 {distance}",
    basis="sto3g",
    charge=0,
    spin=0,
    unit=DistanceUnit.ANGSTROM,
    )
  mapper = ParityMapper() #moguce koristiti i druga preslikavanja
```

```
estimator = Estimator()
es_problem = driver.run()
fermionic_op = es_problem.hamiltonian.second_q_op()
qubit_op = mapper.map(fermionic_op)

# Tocna energija
numpy_solver = NumPyMinimumEigensolver()
result_exact = numpy_solver.compute_minimum_eigenvalue(qubit_op)
exact_energies.append(result_exact.eigenvalue.real +  es_problem.
                                    nuclear_repulsion_energy)

# Postavke za VQE
optimizer = SLSQP(maxiter=80)
initial_state = HartreeFock(
  es_problem.num_spatial_orbitals,
  es_problem.num_particles,
  mapper
  )
ansatz = UCCSD(
  es_problem.num_spatial_orbitals,
  es_problem.num_particles,
  mapper,
  initial_state=initial_state
  )
vqe_solver = VQE(estimator, ansatz, optimizer)
vqe_solver.initial_point = np.zeros(ansatz.num_parameters)
# Koristi GroundStateEigensolver s VQE za dobivanje energije
solver = GroundStateEigensolver(mapper, vqe_solver)
result = solver.solve(es_problem)

vqe_energies.append(result.electronic_energies[0] +   es_problem.
                                    nuclear_repulsion_energy)
```

Konačno grafički prikazujemo rezultate.

```
pylab.plot(distances, exact_energies, 'o-', label="Tocna energija")
pylab.plot(distances, vqe_energies, 'x-', label="VQE energija")
pylab.xlabel("Udaljenost (Angstrom)")
pylab.ylabel("Energija osnovnog stanja (Hartree)")
pylab.title("Energija osnovnog stanja u odnosu na udaljenost")
pylab.legend()
pylab.grid(True)
pylab.show()
```

Iz slike 22 vidimo potpuno poklapanje dobivenih i egzaktnih vrijednosti što je i za očekivati jer se radi o egzaktnoj simulaciji.



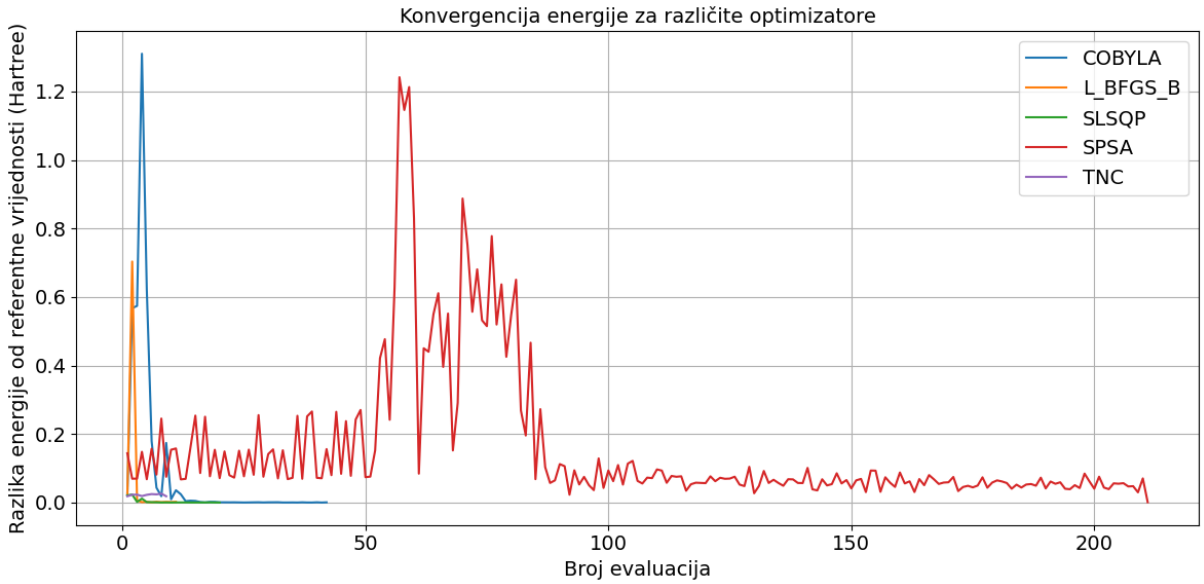
Slika 22: *Graf prikazuje ovisnost energije osnovnog stanja molekule H_2 o udaljenostima među atomima vodika u molekuli. Također je prikazana i energija dobivene klasičnom metodom koju koristimo kao referentu vrijednost. Kako se radi o egzaktnoj simulaciji vidimo odlično slaganje točne energije i VQE energije. Korišten je* `SLSQP` *optimizator.*

5.2.3 Simulacija na idealnom kvantom računalu, usporedba optimizatora

Promotrimo sada simulaciju na idealnom računalu, ponovno ćemo uspoređivati istih pet optimizatora i usporediti njihove rezultate. Prilikom izvođenja simulacije raditi ćemo 5000 mjerenja za jednu aproksimaciju očekivane energije. Kod je potpuno isti kao i za egzaktnu simulaciju osim sljedeće linije koda.

```
estimator = Estimator(options={"shots": 5000})
```

Promotrimo sada rezultate pojedinih optimizatora. Pogledajmo prvo grafičke prikaze rezultata sa slike 23. Uočavamo da optimizatori `SLSQP` i `L_BFGS_B` vrlo brzo konvergiraju prema referentnoj vrijednosti. Iz tablice 8 možemo uočiti da sada svi optimizatori daju bolje rezultate nego što su smo dobivali na idealnom kvantnom računalu za testni hamiltonijan iz prošlog primjera. Jedan od mogućih razloga za to mogu biti inicijalne vrijednosti, budući da je područje pretrage suženo.
Kako bismo se bolje uvjerili koji optimizatori pružaju najbolje rezultate, izvršit ćemo simulaciju više puta pa ćemo promatrati srednju vrijednost energija i njihove standardne devijacije za svaki optimizator. Implementacija ovog koda vrlo je slična onoj za usporedbu optimizatora na idealnom kvantnom računalu. Glavna razlika je što simulaciju pokrećemo unutar petlje 50 puta i koristimo `GroundStateSolver` zbog njegove jednostavnosti.



Slika 23: *Grafički prikaz rezultata različitih optimizatora korištenih u VQE algoritmu za određivanje energije osnovnog stanja molekule H_2 preko simulacije na idealnom kvantom računalu. Na y-osi je prikazana razlika energije od referente vrijednosti, dok je na x-osi prikazan broj evaluacija. Prikazani su* `SPSA`*,* `COBYLA`*,* `L_BFGS_B`*,* `SLSQP` *i* `TNC` *optimizatori. Primijetimo ovdje usporedbom sa slikom 15 gdje je prikazan ovaj isti problem za hamiltonijan $H = 0.7I \otimes X + 0.5I \otimes Y + 0.8X \otimes Z$, kako mnogo bolje rezultate daju svi optimizatori, ovo se događa jer koristimo inicijalne vrijednosti u ansatzu.*

U tablici 9 prikazani su rezultati za srednje vrijednosti i standardne devijacije, pri čemu je za sve optimizatore naveden isti broj znamenki, iako neke nisu značajne. Međutim, na ovaj je način usporedba optimizatora vizualno jednostavnija. Iz tablice 9 uočavamo značajne razlike između pojedinih optimizatora. Iako je u prethodnom primjeru iz Tablice 8 `SLSQP` ispao kao najbolji optimizator, sada vidimo da zapravo daje najgore rezultate s najvećim odstupanjem od referente vrijednosti i najvećom standardnom devijacijom. Kako smo ovdje uvijek imali iste početne vrijednosti dobivene HartreeFock metodom uzrok lošoj izvedbi `SLSQP` optimizatora mogu biti aproksimacije dobivene mjerenjem. Odnosno u slučaju iz tablice 8 `SLSQP` je igrom slučaja izbjegao problem "neplodne visoravni" zbog nasumičnosti u mjerenju. `SPSA` i `COBYLA` se ponovno ističu kao najpouzdaniji optimizatori.

5.2.4 Simulacija na idealnom kvantom računalu, ovisnost energije o udaljenosti među atomima vodika

Razmotrimo simulaciju na idealnom kvantnom računalu gdje istražujemo ovisnost srednje energije o udaljenosti između atoma vodika. Budući da smo iz tablice 9 zaključili da su `SPSA` i `COBYLA` najpouzdaniji optimizatori, možemo upotrijebiti bilo kojeg od njih. Analizirajmo rezultate s `COBYLA` optimizatorom. Iz tablice 10 vidimo da je kemijska preciznost postignuta za 13 udaljenosti od 20. Kod udaljenosti gdje ona nije postignuta odstupanje je vrlo blizu granici preciznosti.

Optimizator	E_{VQE}[Ha]	n_e	$E_{VQE} - E_{REF}$[Ha]
COBYLA	-1.1366988	42	**0.0006072**
L_BFGS_B	-1.1352125	11	0.0020935
SLSQP	-1.1373193	20	**0.0000133**
SPSA	-1.1363327	211	**0.0009733**
TNC	-1.1189765	9	0.0183295

Tablica 8: *Tablica prikazuje rezultate različitih optimizatora korištenih u VQE algoritmu za određivanje energije osnovnog stanja molekule H_2 preko simulacije na idealnom kvantnom računalu. Za svaki optimizator prikazana je konačna vrijednost energije (E_{VQE}), broj evaluacija (n_e), te apsolutna razlika u odnosu na referentnu vrijednost energije ($E_{VQE} - E_{REF}$), gdje referenta energija iznosi $E_{REF} = -1.1373060$Ha. U stupcu $E_{VQE} - E_{REF}$ su podebljani rezultati gdje je postignuta kemijska preciznost.*

Optimizator	E_{AVG}[Ha]	$E_{AVG} - E_{REF}$[Ha]
COBYLA	-1.1362 ± 0.0012	**0.0011**
L_BFGS_B	-1.1166 ± 0.0022	0.0207
SLSQP	-0.3007 ± 0.3603	0.8366
SPSA	-1.1368 ± 0.0004	**0.0005**
TNC	-1.1188 ± 0.0024	0.0185

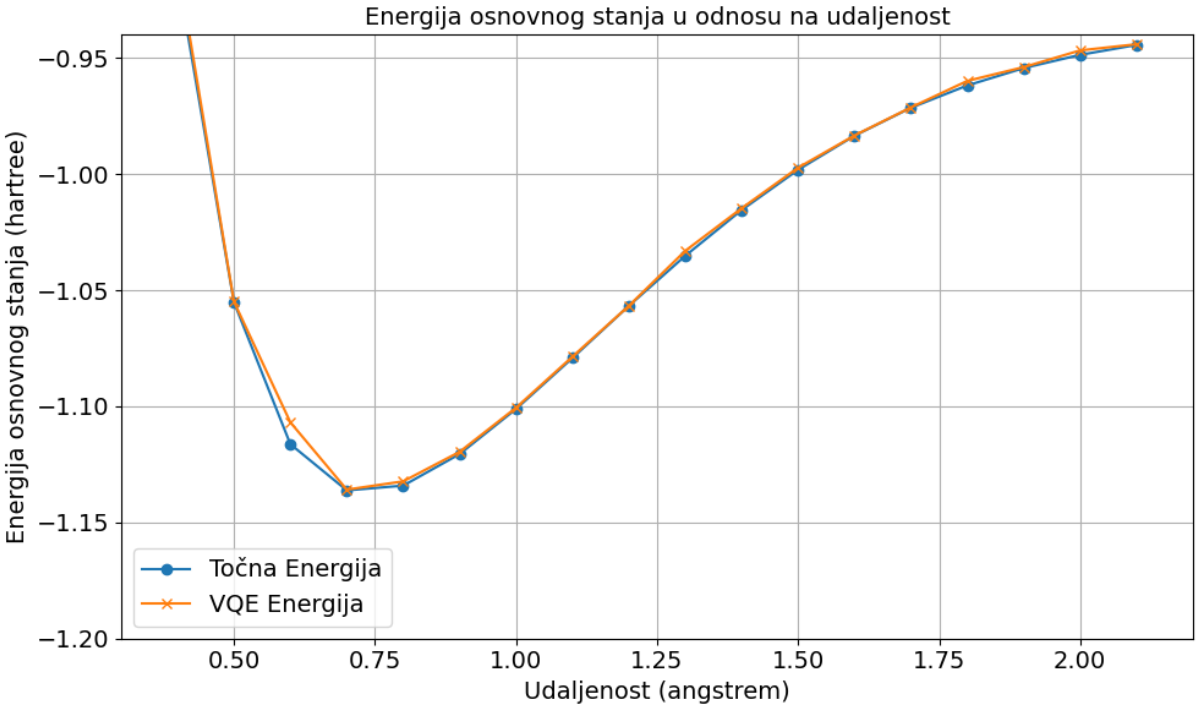
Tablica 9: *Prikaz performansi različitih optimizatora temeljenim na 50 ponavljanja cijele simulacije. Analiza uključuje srednju vrijednost osnovne energije i standardnu devijaciju (E_{VQE}) za svaki optimizator. Osnovne energije su dobivene VQE metodom za molekulu H_2 preko simulacije na idealnom kvantom računalu. Referenta energija iznosi $E_{REF} = -1.1373060$Ha. Primijetimo kako su* `SPSA` *i* `COBYLA` *najpouzdaniji optimizatori. U stupcu $E_{VQE} - E_{REF}$ su podebljani rezultati gdje je postignuta kemijska preciznost.*

5.3 Implementacija za LiH molekulu

Slično kao što smo analizirali molekulu H_2, sada ćemo proučiti molekulu LiH. Razmotrit ćemo egzaktnu simulaciju kao i simulaciju na idealnom kvantnom računalu. Usporediti ćemo koristiti četiri optimizatora: `SPSA`, `COBYLA`, `L_BFGS_B` i `SLSQP`. Odlučili smo izostaviti optimizator `TNC` s obzirom na to da simulacija postaje znatno računalno zahtjevnija s povećanjem složenosti molekule.

5.3.1 Egzaktna simulacija, usporedba optimizatora

Prođimo prvo kroz simulaciju za različite optimizatore, postupak je isti kao i za H_2 molekulu, jedina razlika je u početnoj konfiguraciji molekule.
U ovom primjeru postavljamo udaljenost između atoma na 1.6 Å [32]. Za preslikavanje koristimo `JordanWignerMapper`. Analizirajući dobiveni hamiltonijan, nailazimo na problem koji zahtijeva 12 qubita i sastoji se od 631 tenzorskog produkta. Očigledno je da čak i malo složenija molekula čini problem znatno zahtjevnijim. Budući da ovaj problem rješavamo koristeći lokalno računalo, izvršavanje ove simulacije trajalo bi previše dugo. Kako bi ipak uspjeli



Slika 24: *Graf prikazuje ovisnost energije osnovnog stanja molekule H_2 o udaljenostima među atomima vodika u molekuli. Također je prikazana i energija dobivene klasičnom metodom koju koristimo kao referentu vrijednost. Simulacija je izvršena na idealnom kvantom računalu. Korišten je* `COBYLA` *optimizator.*

izvršiti ovu simulaciju, napraviti ćemo redukciju potrebnog broja qubita.

Qiskit nudi razne opcije koje omogućavaju da preslikavanje reducira broj potrebnih qubita. To se može postići na nekoliko načina, primjerice mogu se promatrati simetrije unutar molekule, mogu se zamrznuti unutarnje ljuske koje ne pridonose mnogo konačnoj energiji. Konkretno, reduciramo broj potrebnih qubita s 12 na 8, čime izvršavanje simulacije postaje smisleno na lokalnom računalu.

Kako bismo reducirali broj qubita koristili smo `TaperedQubitMapper` klasu. Ova klasa radi redukciju tako da prepozna Z_2 simetrije unutar Hamiltonijana i pomoću tih simetrija reducira broj qubita [34]. Ove simetrije ukazuju na to da Hamiltonijan komutira s ovim Paulijevim operatorima, što znači da Hamiltonijan ostaje nepromijenjen kada se ti operatori primijene na njega. Kao rezultat toga, problem se može pojednostaviti uzimajući u obzir ove simetrije. U našem konkretnom primjeru pomoću `tapered_mapper.z2symmetries.symmetries` dobijemo koje simetrije su prepoznate za naš Hamiltonijan: `IIIIIIZIIIII`, `IZIZIIIZIZII`, `IZZIIIIZZIII`, `ZIIIIIIIIIII`.

Također možemo vidjeti pozicijski koji qubite možemo zanemariti. Konkretno, qubiti na pozicijama 5, 2, 3, i 11 (s indeksiranjem od 0) mogu se fiksirati na određene vrijednosti (ili 0 ili 1), što smanjuje veličinu problema [35].

Udaljenost [Å]	E_{VQE}[Ha]	$E_{VQE} - E_{REF}$[Ha]
0.20	0.1579660	**0.0004839**
0.30	-0.5995569	0.0022468
0.40	-0.9092166	0.0049331
0.50	-1.0549172	**0.0002426**
0.60	-1.1068749	0.0094111
0.70	-1.1358662	**0.0003233**
0.80	-1.1322997	0.0018480
0.90	-1.1195874	**0.0009729**
1.00	-1.1005861	**0.0005642**
1.10	-1.0786012	**0.0005917**
1.20	-1.0566984	**0.0000423**
1.30	-1.0330319	0.0021544
1.40	-1.0146807	**0.0007875**
1.50	-0.9972253	**0.0009241**
1.60	-0.9833477	**0.0001250**
1.70	-0.9712694	**0.0001573**
1.80	-0.9598553	0.0019617
1.90	-0.9539124	**0.0004265**
2.00	-0.9467274	0.0019137
2.10	-0.9440695	**0.0003052**

Tablica 10: *Tablica prikazuje energije osnovnog stanja (E_{VQE}) za molekulu H_2 dobivene VQE metodom za različite udaljenosti među atomima vodika u molekuli H_2. Prikazana je razlika između energije dobivenih VQE metodom i energija dobivenih klasičnom metodom (E_{REF}) za svaku pojedinu udaljenost. Primijetimo kako je najniža energija za vrijednost 0.7 Å, što se poklapa s podacima iz literature gdje je najniža energija za udaljenost 0.74 Å. U stupcu $E_{VQE} - E_{REF}$ su podebljani rezultati gdje je postignuta kemijska preciznost.*

Ostatak izvršavanja je isti kao dosada, osim što koristimo objekt `tapered_mapper` umjesto `mapper` objekta.

```
driver = PySCFDriver(
  atom="Li 0 0 0; H 0 0 1.6",
  basis="sto3g",
  charge=0,
  spin=0,
  unit=DistanceUnit.ANGSTROM,
  )
es_problem = driver.run()
mapper = ParityMapper()
fermionic_op = es_problem.hamiltonian.second_q_op()
qubit_op = mapper.map(fermionic_op)
num_qubits_used = qubit_op.num_qubits
print(f"Broj koristenih qubita: {num_qubits_used}") #ovo izbaci 12 qubita,
                                        redukcija jos nije napravljena
#sljedece dvije linije koda rade redukciju
```

```
tapered_mapper = es_problem.get_tapered_mapper(mapper)
qubit_op = tapered_mapper.map(fermionic_op)

#detektirane simetrije
print("Simetrije:")
for symmetry in tapered_mapper.z2symmetries.symmetries:
  print(symmetry)

#koji qubiti se mogu izbaciti
print("\nRedukcija qubita:")
for sq in tapered_mapper.z2symmetries.sq_list:
  print(sq)
```

Pogledajmo sada rezultate ove simulacije. Iz tablice 11 vidimo da `L_BFGS_B` i `SLSQP` daju

Optimizator	E_{VQE}[Ha]	n_e	$E_{VQE} - E_{REF}$[Ha]
COBYLA	-7.8618648	80	0.0204596
L_BFGS_B	-7.8823137	23	0.**0000107**
SLSQP	-7.8823135	23	0.**0000109**
SPSA	-7.0943856	211	0.7879388

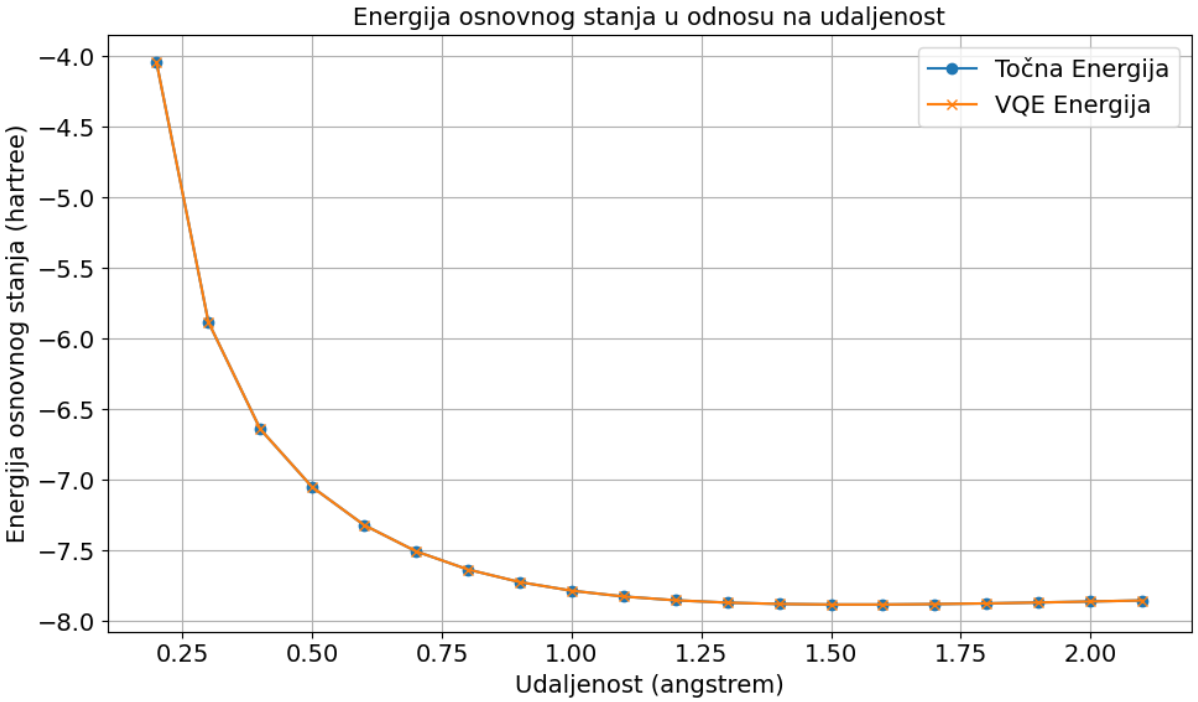
Tablica 11: *Tablica prikazuje rezultate različitih optimizatora korištenih u VQE algoritmu za određivanje energije osnovnog stanja molekule LiH preko egzaktne simulacije. Za svaki optimizator prikazana je konačna vrijednost energije (E_{VQE}), broj evaluacija (n_e), te razlika u odnosu na referentnu vrijednost energije $E_{REF} = -7.8823244$Ha. Primijetimo kako `SPSA` daje lošije rezultate. U stupcu $E_{VQE} - E_{REF}$ su podebljani rezultati gdje je postignuta kemijska preciznost.*

skoro pa točne vrijednosti za egzaktnu simulaciju, oni su i jedini koji su postigli kemijsku preciznost. Ovdje je zanimljivo primijetiti da `SPSA` optimizator slabije optimizira.

5.3.2 Egzaktna simulacija, ovisnost energije o udaljenosti atom litija i vodika

U ovom dijelu analiziramo egzaktnu simulaciju te proučavamo energiju osnovnog stanja u ovisnosti o udaljenosti između atoma u molekuli. Implementacija je identična kao za molekulu vodika, s izuzetkom početne konfiguracije koja je postavljena pomoću klase `PySCFDriver`. Kroz prethodne analize pokazali smo da optimizator `SLSQP` efikasno optimizira za egzaktne simulacije, stoga ćemo ga koristiti i ovdje.

Ponovno vidimo dobro slaganje s pravim vrijednostima.



Slika 25: *Graf prikazuje ovisnost energije osnovnog stanja molekule LiH o udaljenostima među atomima litija i vodika u molekuli. Također je prikazana i energija dobivene klasičnom metodom koju koristimo kao referentu vrijednost. Kako se radi o egzaktnoj simulaciji vidimo odlično slaganje točne energije i VQE energije. Korišten je* `SLSQP` *optimizator.*

5.3.3 Simulacija na idealnom kvantnom računalu, usporedba optimizatora

Da bismo usporedili koji optimizator pruža najbolje rezultate za ovu vrstu simulacije, provodimo simulaciju deset puta preko petlje. Izračunavamo srednje vrijednosti i standardnu devijaciju za svaki pojedini optimizator. Iako je postupak sličan kao za molekulu vodika, ovdje provodimo manje ponavljanja cijele simulacije zbog veće zahtjevnosti problema. U Tablici 11 primjećujemo da su rezultati za molekulu LiH različiti u odnosu na rezultate za molekulu H_2. Iako je `SPSA` prethodno bio jedan od optimizatora koji je često pružao najbolje rezultate za ovu vrstu

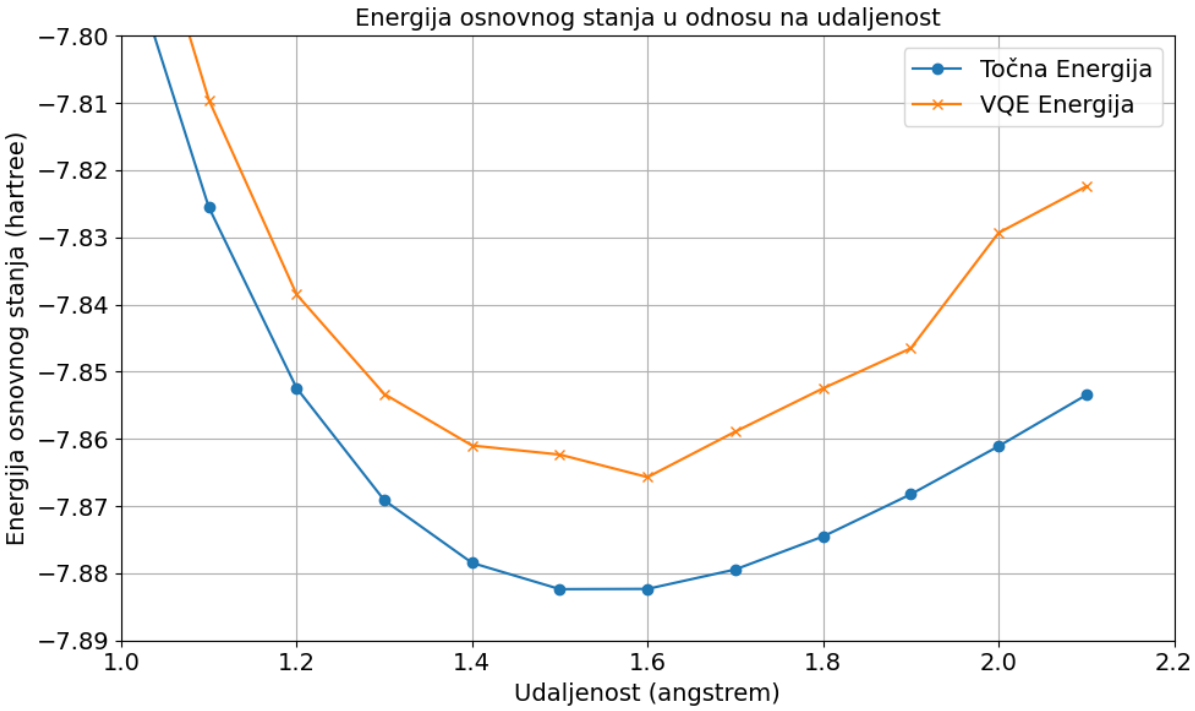
Optimizator	E_{VQE}[Ha]	$E_{VQE} - E_{REF}$[Ha]
COBYLA	-7.8628 $\pm$ 0.0010	0.0195
L_BFGS_B	-7.8622 $\pm$ 0.0015	0.0201
SLSQP	-3.9692 $\pm$ 1.1147	3.9131
SPSA	-6.5208 $\pm$ 0.7251	1.3615

Tablica 12: *Prikaz performansi različitih optimizatora temeljenim na 10 ponavljanja cijele simulacije. Za svaku evaluaciju je rađeno 5000 mjerenja. Analiza uključuje srednju vrijednost energije osnovno stanja LiH molekule i standardnu devijaciju za svaki optimizator (E_{VQE}). Energije su dobivene VQE metodom za molekulu LiH preko simulacije na idealnom kvantom računalu. Prikazana je i razlika u odnosu na referentu energiju ($E_{VQE} - E_{REF}$), gdje je $E_{REF} = -7.8823244$Ha. U stupcu $E_{VQE} - E_{REF}$ su podebljani rezultati gdje je postignuta kemijska preciznost. Primijetimo kako se to nigdje nije dogodilo, unatoč tome* `COBYLA` *i* `L_BFGS_B` *se pokazuju kao najpouzdaniji optimizatori.*

problema, u ovom slučaju daje najgore rezultate. Na temelju ovih saznanja, možemo zaključiti da ne postoji univerzalni optimizator koji će uvijek biti najbolji za svaki problem unutar VQE-a. Preporučljivo je testirati više različitih optimizatora i potom odabrati onaj koji pruža najbolje rezultate za konkretni problem.

5.3.4 Simulacija na idealnom kvantom računalu, ovisnost energije o udaljenosti atoma litija i vodika

Promatrat ćemo ovisnost energije o udaljenosti atoma litija i vodika na simulaciji idealnog kvantnog računala. Za svaku evaluaciju korišteno je 5000 mjerenja. Prema rezultatima iz Tablice 11, `COBYLA` optimizator pruža najbolje rezultate, stoga ćemo ga koristiti za optimizaciju. Budući da je implementacija vrlo slična onoj za molekulu H_2, nećemo prolaziti kroz nju. Iz slike 26 vidimo da se javljaju odstupanja od točne energije. Iz tablice 13 možemo



Slika 26: *Graf prikazuje ovisnost energije osnovnog stanja molekule LiH o udaljenostima među atomima litija i vodika u molekuli. Također je prikazana i energija dobivene klasičnom metodom koju koristimo kao referentu vrijednost. Simulacija je izvršena na idealnom kvantom računalu. Korišten je* `COBYLA` *optimizator.*

i potvrditi da nismo nikada zadovoljili kemijsku preciznost. Možemo i uočiti da se minimum javlja za udaljenost od 1.6 Å za VQE energije, dok klasična metoda daje minimum za udaljenost od 1.5 Å. Ovo se događa jer smo koristili `STO-3G` bazu kod inicijalizacije `PySCFDriver`-a. Korištena baza može biti uzrok manje preciznim rezultatima [36]. Po eksperimentalnoj literaturi udaljenost između atoma litija i vodika iznosi 1.6 Å[32], po ovom ispada da VQE algoritam daje bolje rezultate od klasične metode. Međutim, `STO-3G` baza je bila i korištena za VQE

algoritam tako je ovaj točniji rezultat čista slučajnost.

Udaljenost [Å]	E_{VQE}	$E_{VQE} - E_{REF}$
0.20	-4.0253406	0.0163893
0.30	-5.8584319	0.0233549
0.40	-6.6161989	0.0240789
0.50	-7.0373450	0.0128800
0.60	-7.3013389	0.0179800
0.70	-7.4879895	0.0170623
0.80	-7.6177616	0.0164057
0.90	-7.7065423	0.0168814
1.00	-7.7664069	0.0180534
1.10	-7.8095373	0.0159997
1.20	-7.8384909	0.0139399
1.30	-7.8533225	0.0158175
1.40	-7.8609826	0.0174711
1.50	-7.8623328	0.0200295
1.60	-7.8656880	0.0166364
1.70	-7.8589111	0.0205224
1.80	-7.8524749	0.0220491
1.90	-7.8465117	0.0217291
2.00	-7.8293333	0.0317545
2.10	-7.8224070	0.0310559

Tablica 13: *Tablica prikazuje energije osnovnog stanja (E_{VQE}) za molekulu LiH dobivene VQE metodom za različite udaljenosti među atomima litija i vodika u molekuli. Prikazana je i razlika između energije dobivenih VQE metodom i energija dobivenih klasičnom metodom ($E_{VQE} - E_{REF}$) za svaku pojedinu udaljenost. Primijetimo kako je najniža energija dobivena VQE metodom za vrijednost 1.6* Å*, što se poklapa s udaljenostima iz literatura [32]. Za klasičnu energiju ipak dobivamo da je najniža energija pri 1.5* Å*. Ovdje je energija niža za $-3.79 \cdot 10^5$Ha nego pri 1.6* Å*. Razlog može biti korištenje STO-3G baze u inicijalizaciji PySCF-a koja može biti uzrok loših preciznosti [36].*

6 Zaključak

U ovom radu prvo smo se bavili osnovama kvantnog računanja. Nakon toga, smo se upoznali s Qiskitom, softverskim okvirom razvijenim od strane IBM-a za rad s kvantnim računalima, kroz implementaciju Bellovih stanja i kvantne teleportacije. U nastavku smo se posvetili teorijskim osnovama VQE algoritma, koji koristi varijacijsku metodu za rješavanje svojstvenih problema na kvantnim računalima. Razmotrili smo konstrukciju hamiltonijana te njegovu reprezentaciju u drugoj kvantizaciji. Također smo predstavili različite metode preslikavanja iz fermionskog prostora u prostor Paulijevih operatora, omogućujući prikaz hamiltonijana preko qubita kvantnog računala. Na kraju smo istražili različite ansatze, njihovu konstrukciju te ulogu optimizatora u procesu optimizacije parametara ansatza kako bi energija VQE-a konvergirala prema stvarnoj energiji.

Nakon što smo obradili teorijske osnove, fokusirali smo se na implementaciju VQE-a za jednostavan hamiltonijan $H = 0.7I \otimes X + 0.5I \otimes Y + 0.8X \otimes Z$. U okviru *Qiskita*, implementirali smo simulaciju na četiri različita načina: egzaktnu simulaciju, simulaciju na idealnom kvantnom računalu, simulaciju koja uzima u obzir šumove i simulaciju s mitigacijom grešaka. Analizirali smo izvedbu različitih optimizatora tijekom egzaktne simulacije i simulacije na idealnom kvantnom računalu. Ovu analizu smo napravili za simulacije u kojima smo po jednoj evaluaciji radili 300 mjerenja i 10 000 mjerenja. Utvrdili smo da *SPSA* i *COBYLA* optimizatori pružaju najbolje rezultate s najmanjom standardnom devijacijom za 10 000 mjerenja po evaluaciji. Smanjenjem broja mjerenja na 300, `SPSA` se pokazao kao najbolji optimizator. Osim toga, zaključili smo da različiti ansatzi ne utječu na konačnu energiju za ovaj jednostavan problem. Posebno smo istaknuli važnost mitigacije grešaka: bez nje postižemo točnost od 77.4%, a s njom točnost od 96.7%.

Nakon toga, naša analiza usmjerila se na molekulu vodika. Budući da se radi o problemu iz kvantne kemije, koristili smo kemijsku preciznost za evaluaciju VQE algoritma. To znači da smo provjeravali je li energija osnovnog stanja odstupala za manje od 1.6 mHa od stvarne energije. Izveli smo egzaktnu simulaciju i simulaciju na idealnom kvantnom računalu. U ovoj fazi, problematika se dodatno zakomplicirala jer smo za hamiltonijan dobili sumu od 15 tenzorskih produkata s četiri faktora, što ukazuje na korištenje četiri qubita. Naša analiza je ponovno potvrdila da su *SPSA* i *COBYLA* optimizatori najpouzdaniji. Oni su jedini postigli kemijsku preciznost pri simulaciji na idealnom kvantnom računalu. Također, analizirali smo kako se energija osnovnog stanja mijenja s promjenom udaljenosti između atoma vodika, koristeći *COBYLA* optimizator. Od 20 analiziranih udaljenosti, kemijska preciznost je postignuta 13 puta. Također smo uočili slaganje minimalnih energija u odnosu na udaljenosti s onima iz literature.

Konačno, analizirali smo molekulu LiH. Slično kao i za vodik, izveli smo egzaktnu simulaciju te simulaciju na idealnom kvantnom računalu. Za hamiltonijan ove molekule dobili smo sumu od 631 tenzorskog produkta koji uključuje 12 Paulijevih operatora. Time je izvršavanje

simulacija postalo znatno zahtjevnije, pogotovo jer radimo na klasičnom računalu. Demonstrirali smo mogućnost redukcije broja qubita iskorištavajući simetrije unutar hamiltonijana, čime smo broj korištenih qubita smanjili s 12 na 8. U ovoj analizi, *COBYLA* i *L_BFGS_B* optimizatori pokazali su se kao najučinkovitiji. Nasuprot tome, *SPSA* optimizator, koji je u prethodna dva primjera dao odlične rezultate, u ovom je slučaju pokazao najgore rezultate. Napomenimo kako nijedan optimizator nije postigao kemijsku preciznost. Analizirali smo energije osnovnog stanja LiH u ovisnosti o udaljenosti između atoma u molekuli, te smo uočili da su odstupanja ovdje bila vidljivo veća nego za molekulu vodika.

Usporedbom s rezultatima iz literature [37], vidimo da postoji slaganje s rezultatima kod egzaktnih simulacija. Najlošije rezultate daje *SPSA* optimizator, zatim *COBYLA* optimizator, dok ostala tri optimizatora pokazuju odlične rezultate.

Kod simulacija na idealnom kvantnom računalu, najbolji optimizatori su *COBYLA* i *SPSA*. Ovo je u skladu s literaturom [37], s izuzetkom kod molekule LiH, gdje se *SPSA* pokazao kao najgori. Mogući uzrok ovome može biti mali broj ponavljanja simulacija kod nas. Napomenimo još da je u literaturi korištena maksimalna redukcija qubita za LiH. Time su qubiti korišteni kod LiH svedeni na četiri, dok smo mi koristili osam qubita.

Primjeri koje smo obradili predstavljaju osnovu VQE algoritma. Konstanto se razvijaju nove metode koje unaprjeđuju svaku fazu algoritma. Na primjer, danas postoji velik interes za adaptivne ansatze koji omogućuju varijaciju parametara, čime se smanjuje ukupan broj parametara potrebnih za optimizaciju [3]. Također se razvijaju naprednije verzije ansatza. Mi smo u ovom radu koristili *UCCSD* ansatz, dok se *k-UpCCGSD* (koji je proširenje prethodne metode) pokazuje se kao jedan od najefikasnijih [38]. S obzirom na to da je ovo područje relativno novo, metode se neprestano unaprjeđuju. Kao daljnje poboljšanje koda u ovom radu postoji mogućnost implementacije nekih od ovih naprednijih metoda. Također, postoji i opcija povezivanja s kvantnim računalima koje nudi IBM i izvođenje simulacija na njima. Fokusirali smo se stalno na računanje osnovnog stanje promatranog hamiltonijana, no moguća je implementacija VQE-a i za pobuđenja stanja [39].

7 Literatura

[1] Richard P. Feynman. "Simulating physics with computers". In: *International Journal of Theoretical Physics* 21.6-7 (June 1982), pp. 467–488. DOI: 10.1007/bf02650179.

[2] Jack D Hidary and Jack D Hidary. *Quantum computing: an applied approach*. Vol. 1. Springer, 2019.

[3] Jules Tilly et al. "The variational quantum eigensolver: a review of methods and best practices". In: *Physics Reports* 986 (2022), pp. 1–128.

[4] Michael A Nielsen and Isaac L Chuang. *Quantum computation and quantum information*. Cambridge university press, 2010.

[5] Wikipedia Contributors. *Quantum Computing*. (18. 8. 2023.) URL: https://en.wikipedia.org/wiki/Quantum_computing.

[6] Qiskit Development Team. *Qiskit Documentation*. (18. 8. 2023.) URL: https://qiskit.org/documentation/index.html.

[7] Daniel S. Abrams and Seth Lloyd. "Simulation of Many-Body Fermi Systems on a Universal Quantum Computer". In: *Phys. Rev. Lett.* 79 (13 Sept. 1997), pp. 2586–2589. DOI: 10.1103/PhysRevLett.79.2586.

[8] P. Jordan and E. Wigner. "Über das Paulische Äquivalenzverbot". In: *Zeitschrift für Physik* 47.9-10 (Sept. 1928), pp. 631–651. DOI: 10.1007/bf01331938.

[9] Alexander Altland and Ben D Simons. *Condensed matter field theory*. Cambridge university press, 2010.

[10] Bradben. *Jordan-Wigner Representation - Azure Quantum*. https://learn.microsoft.com/en-us/azure/quantum/user-guide/libraries/chemistry/concepts/jordan-wigner. [Accessed 17-09-2023].

[11] Marco Cerezo et al. "Cost function dependent barren plateaus in shallow parametrized quantum circuits". In: *Nature communications* 12.1 (2021), p. 1791.

[12] Jarrod R McClean et al. "Barren plateaus in quantum neural network training landscapes". In: *Nature communications* 9.1 (2018), p. 4812.

[13] Alexandre Choquette et al. "Quantum-optimal-control-inspired ansatz for variational quantum algorithms". In: *Physical Review Research* 3.2 (2021), p. 023092.

[14] Rodney J Bartlett, Stanislaw A Kucharski, and Jozef Noga. "Alternative coupled-cluster ansätze II. The unitary coupled-cluster method". In: *Chemical physics letters* 155.1 (1989), pp. 133–140.

[15] T Daniel Crawford and Henry F Schaefer III. "An introduction to coupled cluster theory for computational chemists". In: *Reviews in computational chemistry* 14 (2007), pp. 33–136.

[16] Samson Wang et al. “Noise-induced barren plateaus in variational quantum algorithms”. In: *Nature communications* 12.1 (2021), p. 6961.

[17] Ron S Dembo and Trond Steihaug. “Truncated-Newton algorithms for large-scale unconstrained optimization”. In: *Mathematical Programming* 26.2 (1983), pp. 190–212.

[18] Jorge Nocedal and Stephen J Wright. *Numerical optimization*. Springer, 1999.

[19] Richard H Byrd et al. “A limited memory algorithm for bound constrained optimization”. In: *SIAM Journal on scientific computing* 16.5 (1995), pp. 1190–1208.

[20] Michael JD Powell. *A direct search optimization method that models the objective and constraint functions by linear interpolation*. Springer, 1994.

[21] Qiskit Development Team. *Estimator in Qiskit Primitives*. (05. 8. 2023.) URL: https://qiskit.org/documentation/stubs/qiskit.primitives.Estimator.html.

[22] Qiskit Development Team. *Statevector: Qiskit Documentation*. (10. 8. 2023.) URL: https://qiskit.org/documentation/stubs/qiskit.quantum_info.Statevector.html.

[23] Qiskit Development Team. *Ground State Solvers: Qiskit Chemistry Tutorial*. (20. 8. 2023.) URL: https://qiskit.org/documentation/stable/0.25/tutorials/chemistry/03_ground_state_solvers.html.

[24] Qiskit Development Team. *TwoLocal: Qiskit Documentation*. (11. 8. 2023.) URL: https://qiskit.org/documentation/stubs/qiskit.circuit.library.TwoLocal.html.

[25] Kirk A Peterson, David Feller, and David A Dixon. “Chemical accuracy in ab initio thermochemistry and spectroscopy: current strategies and future challenges”. In: *Theoretical Chemistry Accounts* 131 (2012), pp. 1–20.

[26] Jérôme F Gonthier et al. “Measurements as a roadblock to near-term practical quantum advantage in chemistry: resource analysis”. In: *Physical Review Research* 4.3 (2022), p. 033154.

[27] Qiskit Development Team. *Qiskit Aer API Documentation*. (18. 9. 2023.) URL: https://qiskit.org/ecosystem/aer/apidocs/aer.html.

[28] Wikipedia Contributors. *List of Quantum Processors*. (18. 9. 2023.) URL: https://en.wikipedia.org/wiki/List_of_quantum_processors.

[29] Qiskit Development Team. *Qiskit Runtime 0.12.0 documentation: Qiskit Runtime IBM Client Documentation*. (29. 8. 2023.) URL: https://qiskit.org/ecosystem/ibm-runtime/index.html.

[30] Qiskit Development Team. *Configure error mitigation: Qiskit Runtime IBM Client Documentation*. (29. 8. 2023.) URL: https://qiskit.org/ecosystem/ibm-runtime/how_to/error-mitigation.html.

[31] Qiskit Nature Development Team. *PySCFDriver: Qiskit Nature Documentation*. (12. 9. 2023.) URL: https://qiskit.org/ecosystem/nature/stubs/qiskit_nature.second_q.drivers.PySCFDriver.html.

[32] J. Speight. *Lange's Handbook of Chemistry, Seventeenth Edition*. McGraw Hill LLC, 2016. ISBN: 9781259586101.

[33] Qiskit Nature Development Team. *GroundStateSolver: Qiskit Nature Documentation*. (12. 9. 2023.) URL: https://qiskit.org/ecosystem/nature/stubs/qiskit_nature.second_q.algorithms.GroundStateSolver.html.

[34] Qiskit Nature Development Team. *TaperedQubitMapper: Qiskit Nature Documentation*. (10. 8. 2023.) URL: https://qiskit.org/ecosystem/nature/stubs/qiskit_nature.second_q.mappers.TaperedQubitMapper.map.html.

[35] Qiskit Development Team. *Z2Symmetries: Qiskit Documentation*. (12. 9. 2023.) URL: https://qiskit.org/documentation/stubs/qiskit.quantum_info.Z2Symmetries.html.

[36] Cheng-Lin Hong et al. "Accurate and efficient quantum computations of molecular properties using daubechies wavelet molecular orbitals: A benchmark study against experimental data". In: *PRX Quantum* 3.2 (2022), p. 020360.

[37] Harshdeep Singh, Sonjoy Majumder, and Sabyashachi Mishra. "Benchmarking of different optimizers in the variational quantum algorithms for applications in quantum chemistry". In: *The Journal of Chemical Physics* 159.4 (2023).

[38] Dmitry A Fedorov et al. "VQE method: a short survey and recent developments". In: *Materials Theory* 6.1 (2022), pp. 1–21.

[39] Ken M Nakanishi, Kosuke Mitarai, and Keisuke Fujii. "Subspace-search variational quantum eigensolver for excited states". In: *Physical Review Research* 1.3 (2019), p. 033062.