TARTU ÜLIKOOL

Loodus- ja täppisteaduste valdkond

Füüsika instituut

Marko Raidlo

Molekulaarse süsteemi põhioleku energia arvutamine omaväärtusülesande variatsioonilise kvantlahendamise algoritmi abil

Bakalaureusetöö (6 EAP)

Füüsika, keemia ja materjaliteaduse õppekava, füüsika eriala

Juhendajad:

Veiko Palge, PhD

Dirk Oliver Theis, PhD

Tartu 2021

Molekulaarse süsteemi põhioleku energia arvutamine omaväärtusülesande variatsioonilise kvantlahendamise algoritmi abil

Töö käigus rakendati variatsioonilise kvantlahendamise algoritm kvantarvuti simulaatoril molekulide põhiseisundite energiate leidmiseks. Algoritmi kvantahel loodi kasutades unitaarset seotud klastri teooriat ja optimiseerimiseks valiti Nelder-Mead'i meetod. Programmi kvantahelad implementeeriti kasutades kvantarvutamise paketti `cirq` ning simulatsioonid viidi läbi `qsim` olekuvektori simulaatoriga. Molekulaarseteks süsteemideks valiti H_2, LiH ja BeH_2 baasis STO-3G. Kõigi molekulide tasakaalugeomeetria põhiseisundi energiad tulid keemilise täpsuse piires. Muutes sideme pikkuseid kasvas viga suuremaks kui keemiline täpsus.

Märksõnad: kvantarvutamine, kvantkeemia, kvantalgoritmid, variatsiooniline kvantlahendamise algoritm

CERCS: P190 Matemaatiline ja üldine teoreetiline füüsika, klassikaline mehaanika, kvantmehaanika, relatiivsus, gravitatsioon, statistiline füüsika, termodünaamika; P410 Teoreetiline ja kvantkeemia

Calculation of molecular ground state energies using the variational quantum eigensolver algorithm

The variational quantum eigensolver algorithm was implemented on a quantum computer simulator in order to find molecular ground state energies. The creation of the quantum circuit was done using unitary coupled cluster theory and for the optimization the Nelder-Mead method was selected. The quantum circuits were created using the quantum computing package `cirq` and simulated with the `qsim` state vector simulator. The molecular systems selected were H_2, LiH and BeH_2 in minimal STO-3G basis. All ground state energies at equilibrium geometry were found within chemical accuarcy. Ground state energy errors became larger than chemical accuracy with different bond lengths.

Keywords: quantum computation, quantum chemistry, quantum algorithms, variational quantum eigensolver

CERCS: P190 Mathematical and general theoretical physics, classical mechanics, quantum mechanics, relativity, gravitation, statistical physics, thermodynamics; P410 Theoretical chemistry, quantum chemistry

Sisukord

Sissejuhatus

1980ndatel aastatel töötati välja esmane kvantarvutite teooria [1]. Feynmani hüpoteesi kohaselt on kvantarvutitel võimalik efektiivselt simuleerida kvantsüsteeme, kuna need ise on kvantmehaanilised süsteemid [2]. Kaks dekaadi hiljem loodi esimene eksperimentaalne kvantarvuti, millega rakendati Deutsch-Josza algoritmi. Selle eksperimendi käigus suudeti teostada arvutus vähemate funktsiooni väljakutsetega, kui seda oleks võimalik klassikalisel arvutil [3]. Aastal 2019 jõuti järgmise suure sammuni kvantarvutites kui Google teatas, et on saavutanud "quantum supremacy", rakendades kvantarvutil algoritmi, mis on kvantarvutil kordades kiirem kui sama arvutus klassikalisel arvutil [4]. Kuigi Google suutis eksperimentaalselt näidata kvantarvutite eelist klassikaliste arvutite ees, pole rakendatud algoritm eriti kasulik. Järgmine kvantarvutite suur samm on kvantarvutil kasuliku arvutuse läbi viimine kiiremini kui seda suudab teha klassikaline arvuti.

Molekulaarsete süsteemide simuleerimine on üks tähtsamatest kvantarvuti rakendustest krüptograafia ja masinõppe kõrval. Molekulide väga täpsete energiate arvutamine võib aidata kaasa uute materjalide ning ravimite sünteesimisele. Kahjuks tänapäeva kvantarvutitel on mitu probleemi, mis takistavad suurte kvantsüsteemide simuleerimist. Esiteks müra rikub kvantbittide olekuid, mistõttu ei püsi need piisavalt kaua stabiilsena, et oleks võimalik pikki kvantahelaid rakendada. Lisaks iga kvantvärava rakendus kaasneb väikese veaga. Suur väravate hulk kaotab kogu kasuliku info enne mõõtmist ära. Nende probleemide lahendusteks on välja töödatud erinevad optimiseerimismeetodid ning veaparanduskoodid. Veaparanduskoodid salvestavad kvantbiti informatsioon mitmesse kvantbitti, selleks et vähendada müra ning üksikute väravate vigade mõju kvantolekule. Veaparanduskoodide rakendamiseks on aga vaja miljoneid füüsilisi kvantbitte, mistõttu on nende rakendamine kauges tulevikus, sest töö kirjutamise hetkel on suurim kvantarvuti 76 kvantbitiga [5].

Selleks, et kvantsüsteemi simuleerimine oleks ka võimalik praegustel lähiaja väikese kvantbittide arvuga vigadega masinatel (ingl NISQ, Noisy Intermediate-Scale Quantum) on loodud omaväärtusülesande variatsioonilise kvantlahendamise algoritm (ingl VQE, variational quantum eigensolver). VQE algoritm kasutab ühe pika raskesti implementeeritava kvantahela asemel mitmeid väikseid, millel tehakse vähem mõõtmisi. Lisaks ei vaja VQE algoritm veaparanduskoode. Selle tagajärjel suudab VQE algoritm arvutada väikeste molekulide energiaid juba

praegu olemasolevatel kvantarvutitel [6].

Antud töö eesmärgiks on rakendada VQE algoritm ning arvutada H_2, LiH ja BeH_2 molekulide põhioleku ning erinevate geomeetriate energiad kvantarvuti simulaatoril. Molekulid valiti nende lihtsuse tõttu. Suurem elektronide arv molekulides kasvatab vajaminevat kvantbittide arvu liiga suureks simulaatori jaoks.

Töö esimene peatükk käsitleb elektronstruktuuri probleemi, üldist kvantarvutamist, VQE algoritmi, algoritmi jaoks kvantahela loomist ning selle mõõtmishamiltoniaani leidmist. Teine peatükk kirjeldab arvutuste läbiviimiseks loodud programmi tööd. Kolmandas peatükis tuuakse välja arvutuste tulemused ning nende analüüs. Lisades A.1 ja A.2 on toodud arvutuste läbiviimiseks kirjutatud programmide `vqe_functions.py` ja `run_vqe.py` kood.

1 Teoreetiline taust

1.1 Elektronstruktuuri probleem

Kvantkeemia on keemia haru, mis kasutab kvantmehaanikat aatomite ja molekulide struktuuri kirjeldamiseks. Kuna nende keemilised omadused on tingitud elektronkattest, siis on kvantkeemias olulisel kohal elektronstruktuuri kirjeldamine. Molekulide jaoks saab kirja panna statsionaarse Schrödingeri võrrandi kujul

$$H\Psi = E\Psi, \tag{1.1}$$

kus H on molekulaarse süsteemi hamiltoniaan, Ψ on lainefunktsioon ning E on energia omaväärtus. Elektronstruktuuri probleemis on eesmärgiks leida energia omaväärtused E, sest läbi nende on määratud enamus molekuli omadustest. N elektroni ja M tuumaga molekulaarse süsteemi hamiltoniaan on antud kujul

$$H = -\sum_{i=1}^{N} \frac{1}{2}\nabla_i^2 - \sum_{A=1}^{M} \frac{1}{2M_A}\nabla_A^2 - \sum_{i=1}^{N}\sum_{A=1}^{M} \frac{Z_A}{r_{iA}} + \sum_{i=1}^{N}\sum_{j>i}^{N} \frac{1}{r_{ij}} + \sum_{A=1}^{M}\sum_{B>A}^{M} \frac{Z_A Z_B}{R_{AB}}, \tag{1.2}$$

kus esimene liige vastab elektronide kineetilisele energiale, teine tuumade kineetilisele energiale, kolmas elektrostaatilise vastastikmõju energiale elektronide ja tuumade vahel ja neljas ning viies vastastikmõju energiale vastavalt elektronide ja tuumade vahel. Antud hamiltoniaani omaväärtuste leidmiseks käesolevas töös kasutatakse omaväärtusülesande variatsioonilise kvantlahendamise algoritmi (ingl VQE, variational quantum eigensolver).

Järgnevas alapeatükis anname lühiülevaate kvantarvutite väravatest ning ahelatest. Kolmandas alapeatükis defineerimie VQE algoritmi tööpõhimõtte. Neljandas alapeatükis toome välja, kuidas kujutada molekulaarse süsteemi lainefunktsiooni kvantarvutil unitaarse operaatori abil. Viimases alapeatükis kirjeldame, kuidas esitada avaldis (1.2) kujul, mida on võimalik kvantarvutil arvutamiseks rakendada.

1.2 Kvantarvutamine

1.2.1 Kvantbitt

Kvantarvuti kõige fundamentaalsem osa on kvantbitt. Sarnaselt klassikalise arvuti bitile, mis võib olla 0 või 1, võib kvantbitt olla olekus $|0\rangle$ ja $|1\rangle$. Erinevus biti ja kvantbiti vahel seisneb selles, et kvantbitt võib olla ka superpositsioonis kahest võimalikus olekust

$$|\Psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle, \tag{1.3}$$

kus koefitsiendid α ja β on kompleksarvud ning $|\Psi\rangle$ on olekuvektor [7]. Kvantbiti olekuvektor on ühikvektor, seega koefitsiendid peavad järgima normeerituse tingimust

$$|\alpha|^2 + |\beta|^2 = 1. \tag{1.4}$$

Superpositsioonis olev kvantbitt kollapseerub mõõtmise hetkel ühte kahest baasiolekust, tõenäosustega $|\alpha|^2$ ja $|\beta|^2$ [8].

Kahe kvantbitisel süsteemil on neli baasolekut $|00\rangle$, $|01\rangle$, $|10\rangle$ ja $|11\rangle$ ning süsteemi seisundit saab avaldada nende lineaarse kombinatsioonina

$$|\varphi\rangle = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle. \tag{1.5}$$

N kvantbitist kvantsüsteemi saab kirjeldada olekuvektoriga, mille olek avaldub 2^N baasivektori lineaarse kombinatsioonina [7]. Sellest on tingitud kvantarvutite eelis klassikaliste arvutite ees. Kvantbittide arvu kasvades suureneb eksponentsiaalselt informatsioon, mida selle süsteemi kohta klassikaline arvuti mälus talletama peab. Kvantarvutil antud probleemi ei teki.

1.2.2 Kvantväravad

Kvantbittide ühest olekust teise viimiseks kasutatakse kvantväravaid. Kvantväravad on operaatorid, mis mõjuvad kvantbiti olekuvektorile. Väravad võivad mõjuda ühele või mitmele kvantbitile. Näiteks NOT ehk X kvantvärav muudab kvantbiti ühest baasiolekust teise $X|0\rangle = |1\rangle$ ja $X|1\rangle = |0\rangle$. Superpositsioonile mõjudes vahetuvad baasivektorite koefitsiendid [7]

$$X|\Psi\rangle = X(\alpha|0\rangle + \beta|1\rangle) = \alpha|1\rangle + \beta|0\rangle. \tag{1.6}$$

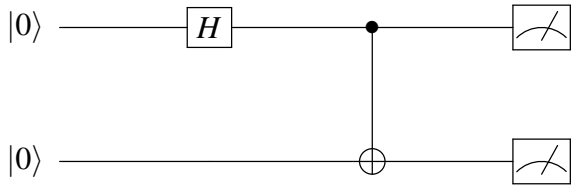
Kvantväravaid on võimalik kujutada maatriksina. Järgmisena on välja toodud näitena mõned levinud kvantväravad.

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad R_z(\theta) = \begin{pmatrix} \cos\frac{\theta}{2} & -i\sin\frac{\theta}{2} \\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix}, \quad CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{1.7}$$

$R_x(\theta)$ kasutatakse kvantbiti olekuvektori pööramiseks θ võrra ümber X operaatori omavektorite. CNOT värav mõjub kahel kvantbitil. Kui kontroll-kvantbitt on olekus $|1\rangle$, siis mõjub X värav siht-kvantbitile. Juhul kui kontroll-kvantbitt on olekus $|0\rangle$, siis siht-kvantbitt jääb samasse olekusse. Antud väravaga on võimalik kvantbitte omavahel põimida.

1.2.3 Kvantahelad

Kvantahelad koosnevad kvantbittidest ning kvantväravatest. Igat kvantbitti tähistab horisontaalne "traat" ning mööda seda võib kujutada kvantbitti liikumas ajas. Igasse kvantväravasse siseneb ning väljub nii palju traate, kui mitmele kvantbitile antud värav mõjub. Kvantväravad on asetatud ajalises järjestuses [8]. Alghetkel on kvantahela kõik kvantbitid algolekus, mis on tavaliselt $|0\rangle$. Kvantahela lõpus vajalikud kvantbitid mõõdetakse, seda tähistakse mõõteriista sümboliga.



Joonis 1.1: Näide kvantahelast. Alghetkel on kaks kvantbitti algolekus $|0\rangle$. Järgnevalt mõjub Hadamardi värav esimesele kvantbitile. Edasi mõjub mõlemale CNOT värav, pärast mida mõlemad kvantbitid mõõdetakse.

1.3 Omaväärtusülesande variatsioonilise kvantlahendamise algoritm

1.3.1 Variatsiooniprintsiip

Võtame süsteemi, mida iseloomustab hamiltoniaan H ning leiame põhioleku energia E_p. Kui süsteem on piisavalt keeruline, ei ole võimalik analüütiliselt lahendada Schrödingeri võrrandit ning tuleb energia leidmiseks kasutada lähendust. Kui $|\Psi\rangle$ on suvaline normeeritud olekuvektor,

siis variatsiooniprintsiibi kohaselt

$$\langle H \rangle \equiv \langle \Psi | H | \Psi \rangle \geq E_p. \tag{1.8}$$

Iga seisundi $|\Psi\rangle$ korral on hamiltoniaani keskväärtus suurem või võrdne energia põhiseisundiga. Sobiva $|\Psi\rangle$ valimisel on võimalik leida hea lähendusega põhioleku energia väärtus [9].

1.3.2 VQE algoritm

Omaväärtusülesande variatsiooniline kvantlahendamine on algoritm, mis rakendab variatsiooniprintsiipi kvantarvutil hea lähendusega molekuraase süsteemi põhioleku energia leidmiseks. Olgu kvantarvuti N kvantbitiga ja kvantsüsteem hamiltoniaaniga H. VQE algoritmiga on võimalik leida hamiltoniaani H väikseim omaväärtus λ_i. Olgu rida parameetreid θ_i, mida saab esitada vektorina $\vec{\theta}$. Kui kvantarvuti seada olekusse, mis sõltub nendest parameetritest, $|\Psi(\vec{\theta})\rangle$, siis variatsiooniprintsiibi kohaselt

$$\langle H \rangle_{|\Psi(\vec{\theta})\rangle} \equiv \langle \Psi(\vec{\theta}) | H | \Psi(\vec{\theta}) \rangle \geq \lambda_1. \tag{1.9}$$

Parameetrite $\vec{\theta}$ optimiseerimisega on võimalik keskväärtust $\langle H \rangle_{\Psi(\vec{\theta})}$ minimiseerida ehk leida põhioleku energia.

VQE algoritmi rakendamine jaguneb järgmisteks sammudeks:

1. Kvantarvutil valmistatakse ette seisund $|\Psi(\vec{\theta})\rangle$.
2. Mõõdetakse selle seisundi keskväärtus $\langle H \rangle(\vec{\theta})$.
3. Kasutades leitud keskväärtust leitakse optimeerimise teel uued $\vec{\theta}$ väärtused, mis vähendavad hamiltoniaani keskväärtust.
4. Uute $\vec{\theta}$ väärtustega korratakse protsessi kuni toimub koondumine miinimumi, milleks ongi uuritava süsteemi põhioleku energia [10].

1.4 Unitaarne seotud klastri lainefunktsioon

1.4.1 Molekulaarorbitaalide teooria

Molekulaarorbitaalide teooria kirjeldab elektronide asetust molekulides. Erinevalt lihtsamatest keemiliste sidemete teooriatest, kus elektrone kujutatakse molekulis kindlates sidemetes aatomite vahel, on molekulaarorbitaalides elektronide lainefunktsioon jaotunud kõigi molekulis ole-

vate aatomite vahel. Molekulaarobitaal moodustatakse lineaarkombinatsioonina aatomorbitaalidest. Näiteks vesiniku molekuli molekulaarorbitaalid moodustuvad kahest 1s aatomorbitaalist

$$\Psi_{\pm} = \psi_{A1s} \pm \psi_{B1s}. \tag{1.10}$$

Molekulaarorbitaalid jagatakse kolmeks eri tüübiks vastavalt orbitaali energiale. Näiteks, kui vesiniku kaks aatomorbitaali on samamärgilised, siis neid liites molekulaarorbitaaliks tekib kahe aatomi vahele lainefunktsioonide interferentsimaksimum. Kuna elektroni lainefunktsioon antud molekulaarorbitaalil on jaotatud suurema ruumala peale kui üksiku aatomi juures, on elektroni energia väiksem. Energiat vähendavaid orbitaale nimetatakse siduvateks orbitaalideks.

Kui vesiniku näitel kaks aatomorbitaali on vastasmärgilised, tekib kahe aatomi vahele interferentsimiinimum, kus kaks elektroni lainefunktsiooni kustutavad teineteist. Molekulaarorbitaalil olev elektron ei saa antud miinimumis eksisteerida ja järelikult on elektron väiksemas ruumalas ning energia on suurem. Energiat suurendavaid orbitaale nimetatakse lõdvendavateks orbitaalideks. Suure aatomite arvuga molekulides, kus on komplekssemad molekulaarorbitaalid, esinevad ka mittesiduvad orbitaalid, mis ei panusta sidemete sidumisse ega lõdvendamisse. K aatomorbitaali kohta tekib K molekulaarorbitaali [11].

Molekulaarorbitaalide elektronide jaotust aatomist saab kirjeldada ruumiorbitaaliga $\psi_i(\boldsymbol{r})$, kus vektor $\boldsymbol{r}$ on kolmedimensionaalne ruumivektor. Ruumiorbitaali tõenäosustihedus $|\psi_i(\boldsymbol{r})|^2 d\boldsymbol{r}$ kirjeldab tõenäosust leida elektron punkti $\boldsymbol{r}$ ümbruses $d\boldsymbol{r}$.

Elektroni täielikuks kirjeldamiseks on lisaks asukohale ruumis vaja lisada spinn. Spinni on võimalik defineerida kahe normaalfunktsiooniga $\alpha(\omega)$ ja $\beta(\omega)$, mis vastavad spinn üles ning spinn alla olekutele. Spinnorbitaal $\chi(\boldsymbol{x})$ on lainefunktsioon, mis määrab ära elektroni asukoha ning spinni molekulis. $\boldsymbol{x}$ on vektor, mis koosneb ruumivektorist $\boldsymbol{r}$ ja komponendist, mis määrab spinni oleku. Iga ruumiorbitaali kohta tekib kaks spinnorbitaali

$$\chi(\boldsymbol{x}) = \begin{cases} \psi(\boldsymbol{r})\alpha(\omega) \\ \psi(\boldsymbol{r})\beta(\omega). \end{cases} \tag{1.11}$$

K molekulaarorbitaali puhul saab defineerida $2K$ spinnorbitaali, see tähendab igal elektronil on kaks võimalikku spinnolekut [12].

1.4.2 Hartree produkt ja Slateri determinant

Järgmiseks käsitleme mitme elektroni lainefunktsiooni. Oletame, et N mitte-interakteeruva elektronilise süsteemi hamiltoniaan on antud kujul

$$H = \sum_{i=1}^{N} h(i), \tag{1.12}$$

kus $h(i)$ on operaator, mis kirjeldab i-nda elektroni energiat. Igale operaatorile $h(i)$ vastab omaväärtus ε_j

$$h(i)\chi_j(\boldsymbol{x}_i) = \varepsilon_j \chi_j(\boldsymbol{x}_i). \tag{1.13}$$

Kuna süsteemi hamiltoniaan on summa eri elektronide operaatoridest, on süsteemi lainefunktsioon Ψ^{HP} korrutis elektronide lainefunktsioonidest

$$\Psi^{HP}(\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_N) = \chi_i(\boldsymbol{x}_1)\chi_j(\boldsymbol{x}_2)...\chi_k(\boldsymbol{x}_N). \tag{1.14}$$

Saadud lainefunktsioon on hamiltoniaani omafunktsioon

$$H\Psi^{HP} = E\Psi^{HP} \tag{1.15}$$

ning kogu süsteemi energia omaväärtus on

$$E = \varepsilon_i + \varepsilon_j + ... + \varepsilon_k. \tag{1.16}$$

Sellist mitme elektroni lainefunktsiooni nimetatakse Hartree produktiks. Hartree produkt ei rahulda antisümmeetria printsiipi, mille kohaselt mitmefermionilise süsteemi lainefunktsioon peab osakeste vahetamisel muutma märki. Seda puudust saab likvideerida, kui võtta lineaarne kombinatsioon Hartree produktidest. Kahe-elektronilise antisümmeetrilise lainefunktsiooni saab esitada kujul

$$\Psi(\boldsymbol{x}_1, \boldsymbol{x}_2) = \frac{1}{\sqrt{2}} \left(\chi_i(\boldsymbol{x}_1)\chi_j(\boldsymbol{x}_2) - \chi_j(\boldsymbol{x}_1)\chi_i(\boldsymbol{x}_2) \right), \tag{1.17}$$

kus $1/\sqrt{2}$ on normeering. Antud kombinatsiooni saab ümber kirjutada Slateri determinandina

$$\Psi(\boldsymbol{x}_1, \boldsymbol{x}_2) = \frac{1}{\sqrt{2}} \begin{vmatrix} \chi_i(\boldsymbol{x}_1) & \chi_j(\boldsymbol{x}_1) \\ \chi_i(\boldsymbol{x}_2) & \chi_j(\boldsymbol{x}_2) \end{vmatrix}. \tag{1.18}$$

N elektronilise süsteemi jaoks on Slateri determinant antud kujul

$$\Psi(\boldsymbol{x}_1,\boldsymbol{x}_2,...,\boldsymbol{x}_N) = \frac{1}{\sqrt{N!}}\begin{vmatrix} \chi_i(\boldsymbol{x}_1) & \chi_j(\boldsymbol{x}_1) & ... & \chi_k(\boldsymbol{x}_1) \\ \chi_i(\boldsymbol{x}_2) & \chi_j(\boldsymbol{x}_2) & ... & \chi_k(\boldsymbol{x}_2) \\ ... & ... & ... & ... \\ \chi_i(\boldsymbol{x}_N) & \chi_j(\boldsymbol{x}_N) & ... & \chi_k(\boldsymbol{x}_N) \end{vmatrix}, \tag{1.19}$$

kus igale reale vastab elektron ning igale veerule vastab spinnorbitaal. Edasises käsituses tähistame Hartree produkti Slateri determinandi märkides ainult diagonaali spinnorbitaalid, kus spinnorbitaalil χ_i asetseb elektron $\boldsymbol{x}_1$

$$|\chi_i(\boldsymbol{x}_1),\chi_j(\boldsymbol{x}_2),...,\chi_k(\boldsymbol{x}_N)\rangle = |\chi_i\chi_j...\chi_k\rangle. \tag{1.20}$$

1.4.3 Hartree-Focki lähendus

Hartree-Focki lähenduse põhieesmärk on lihtsustada elektronide interaktsioonide käsitlust. Lähenduse jaoks keskmistatakse iga elektroni jaoks kõikide teiste elektronide mõju ühtseks potentsiaaliks. Hartree-Fock lähenduses on igale elektronile seatud vastavusse Focki operaator

$$f(i) = -\frac{1}{2}\nabla_i^2 - \sum_{A=1}^{M}\frac{Z_A}{r_{iA}} + v^{HF}, \tag{1.21}$$

kus v^{HF} kirjeldab keskmist potentsiaali, mis mõjub elektronile i teiste elektronide toimel. Hartree-Focki lähenduses on iga Focki operaatori omaväärtus vastava elektroni spinnorbitaali energia ε. Hartree-Focki omaväärtusvõrrand on

$$f(i)\chi(\boldsymbol{x}_i) = \varepsilon\chi(\boldsymbol{x}_i). \tag{1.22}$$

Antud võrrandi lahendamise kaudu leitakse Hartree-Focki (HF) põhioleku lainefunktsioon

$$|\Psi_0\rangle = |\chi_1\chi_2...\chi_N\rangle, \tag{1.23}$$

kus elektronidega on täidetud ainult madalama energiatasemega spinnorbitaalid, mida nimetatakse *täidetud* orbitaalideks. Antud lahenduse käigus leitakse ka $2K-N$ spinnorbitaali, mis ei ole täidetud põhiolekus. Neid spinnorbitaale nimetatakse *virtuaalseteks* orbitaalideks.

Lisaks HF põhiolekule on suur hulk teisi konfiguratsioone, milles antud süsteem võib olla. Elektronid võivad olla ergastatud oma põhioleku spinnorbitaalist mõnele virtuaalsele spinnorbitaalile. Näiteks kui elektron, mis asus spinnorbitaalil χ_x ergastati virtuaalsele spinnorbitaalile

χ_y, saab lainefunktsiooni anda kujul

$$|\Psi_x^y\rangle = |\chi_1\chi_2\ldots\chi_y\ldots\chi_N\rangle. \tag{1.24}$$

Korraga võib olla ergastatud kuni N elektroni. HF põhiolekust ja kõigist võimalikest ergastatud olekutest on võimalik moodustada kogu süsteemi olekut kirjeldav lainefunktsioon

$$|\Phi\rangle = c_0|\Psi_0\rangle + \sum_{xy} c_x^y|\Psi_x^y\rangle + \sum_{x<z,y<w} c_{xz}^{yw}|\Psi_{xz}^{yw}\rangle + \ldots . \tag{1.25}$$

Antud täieliku konfiguratsiooni vastastikmõju (ingl FCI, full configuration interaction) lainefunktsioon koosneb kõigist võimalikest konfiguratsioonidest kuni selleni, et kõik virtuaalsed spinnorbitaalid on täidetud [12]. Arvutuste jaoks ei ole võimalik kõiki konfiguratsioone kasutada nende suure arvu tõttu. Antud töös käsitleme ainult ühe- ja kahekordseid ergastusi

$$|\Phi\rangle = c_0|\Psi_0\rangle + \sum_{xy} c_x^y|\Psi_x^y\rangle + \sum_{x<z,y<w} c_{xz}^{yw}|\Psi_{xz}^{yw}\rangle. \tag{1.26}$$

1.4.4 Teine kvantiseerimine

FCI lainefunktsioonis saime iga konfiguratsiooni jaoks erineva lainefunktsiooni. Kasutades teist kvantiseerimist kirjeldame igat konfiguratsiooni põhioleku lainefunktsiooni kaudu. Selleks defineerime tekke- ja kaooperaatorid. Tekkeoperaator $a_i^\dagger$ tekitab süsteemi juurde elektroni spinnorbitaalil χ_i

$$a_i^\dagger|\chi_j\ldots\chi_N\rangle = |\chi_i\chi_j\ldots\chi_k\rangle. \tag{1.27}$$

Kaooperaator a_i eemaldab süsteemis elektroni spinnorbitaalil χ_i [12]

$$a_i|\chi_i\chi_j\ldots\chi_k\rangle = |\chi_j\ldots\chi_k\rangle. \tag{1.28}$$

Tekke- ja kaooperaatorite algebralised omadused on määratud nende antikommutatsiooni reeglitega

$$[a_i^\dagger, a_j^\dagger]_+ = 0, \quad [a_i, a_j]_+ = 0, \quad [a_i^\dagger, a_j]_+ = \delta_{i,j}. \tag{1.29}$$

Antud kahe operaatoriga on võimalik kõiki FCI konfiguratsioone esitada süsteemi põhioleku $|\Psi_0\rangle$ kaudu. Võtame näiteks N elektronilise süsteemi, kus üks elektron on ergastatud põhioleku orbitaalilt χ_i virtuaalsele spinnorbitaalile χ_x

$$|\Psi_i^x\rangle = |\chi_1\chi_2\ldots\chi_x\ldots\chi_N\rangle. \tag{1.30}$$

Alustuseks võtame põhioleku $|\Psi_0\rangle$ ja rakendame talle kaooperaatorit a_i, mille tulemusena saame seisundi

$$a_i|\Psi_0\rangle = a_i|\chi_1\chi_2...\chi_i...\chi_N\rangle = (-1)^{i-1}|\chi_1\chi_2...\chi_N\rangle. \tag{1.31}$$

Järgmiseks rakendame tekkeoperaatorit $a_x^\dagger$, mis tekitab χ_x virtuaalsele orbitaalile elektroni

$$a_x^\dagger|\chi_1\chi_2...\chi_N\rangle = (-1)^{x-1}|\chi_1\chi_2...\chi_x...\chi_N\rangle. \tag{1.32}$$

Nüüd saame esitada konfiguratsiooni $|\Psi_i^x\rangle$ teises kvantiseerimises tekke- ja kaooperaatorite kaudu

$$|\Psi_i^x\rangle = C_i^x a_x^\dagger a_i|\Psi_0\rangle, \tag{1.33}$$

kus C_i^x sisaldab tekke- ja kaooperaatorite faasimuutusi ning FCI koefitsenti c_i^x. Tekke- ja kaooperaatorite abil saab defineerida ergastuseoperaatori $X_{i..,j}^{k...l}$, mis tõstab elektronid orbitaalidelt $i,...,j$ orbitaalidele $k,...,l$. Valem (1.33) võtab kuju

$$|\Psi_i^x\rangle = X_i^x|\Psi_0\rangle. \tag{1.34}$$

Ergastusoperaatoritega saab FCI lainefunktsiooni esitada kujul [13]

$$|\Phi\rangle = (1+\sum_{xy} X_x^y + \sum_{x<z,y<w} X_{xz}^{yw} + ...)|\Psi_0\rangle. \tag{1.35}$$

1.4.5 Seotud klastri teooria

Lainefunktsiooni arvutuste jaoks paremale kujule viimiseks toome sisse seotud klastri (ingl CC, coupled cluster) teooria. CC teooria järgi saab FCI lainefunktsiooni esitada kujul

$$|CC\rangle = \left[\prod_{xy}(1+X_x^y)\right]\left[\prod_{x<z,y<w}(1+X_{xz}^{yw})\right]...|\Psi_0\rangle. \tag{1.36}$$

CC lainefunktsiooni on võimalik lisaks produktile esitada eksponendina. Kuna vastavalt tekke- ja kaooperaatorite omadustele kehtib

$$X_{xz}^{yw}X_{xz}^{yw} = 0, \tag{1.37}$$

saab kirjutada MacLaurini seeria abil

$$1+X_{xz}^{yw} = 1+X_{xz}^{yw}+\frac{1}{2}X_{xz}^{yw}X_{xz}^{yw}+... = \exp\left(X_{xz}^{yw}\right). \tag{1.38}$$

Sama loogikat saab rakendada kogu CC lainefunktsioonile

$$|CC\rangle = \exp\left(\sum_{xy} X_x^y + \sum_{x<z,y<w} X_{xz}^{yw} + ...\right)|\Psi_0\rangle. \tag{1.39}$$

Tähistades klasteroperaatori sümboliga T ja defineerides

$$T_1 = \sum_{xy} t_y^x a_x^\dagger a_y, \quad T_2 = \frac{1}{4}\sum_{xzyw} t_{xz}^{yw} a_y^\dagger a_w^\dagger a_x a_z \tag{1.40}$$

saame CC lainefunktsiooni viia kujule

$$|CC\rangle = \exp(T)|\Psi_0\rangle, \tag{1.41}$$

kus $T = T_1 + T_2 + ...$. Lainefunktsioon ainult üksik- ja topeltergastuste jaoks on antud kujul (ingl CCSD, couple cluster single double) [13]

$$|CCSD\rangle = \exp(T_1 + T_2)|\Psi_0\rangle. \tag{1.42}$$

Kvantarvutil rakendamiseks peab operaator olema unitaarne. Unitaarsuse saavutab kaasoperaatori lahutamine ergastuse operaatorist, selle tulemusena saame unitaarse seotud klastri operaatori (ingl UCCSD, unitary couple cluster single double)

$$|UCCSD\rangle = \exp(T - T^\dagger)|\Psi_0\rangle. \tag{1.43}$$

1.4.6 Kvantahela loomine

Viimase sammuna on vajalik UCCSD lainefunktsioon viia kujule, mida saab rakendada kvantarvutil. Esmalt on vaja fermionoperaatorid teisendada kvantbittoperaatoriteks. Seda saavutame kasutades Jordan-Wigneri teisendust. Jordan-Wigneri teisenduse tulemusena iga kvantbitt tähistab 0 või 1-ga, kas sellele vastav spinnorbitaal on täidetud elektroniga. Kao- ja tekkeoperaatorite teisendus on antud kujutistega

$$a_p^\dagger \mapsto Q_p^\dagger \otimes Z_{p-1} \otimes ... \otimes Z_0, \tag{1.44}$$

$$a_p \mapsto Q_p \otimes Z_{p-1} \otimes ... \otimes Z_0, \tag{1.45}$$

kus $Q^\dagger = |1\rangle\langle 0| = \frac{1}{2}(X - iY)$ ja $Q = |0\rangle\langle 1| = \frac{1}{2}(X + iY)$. Jordan-Wigneri teisenduse tulemusena saab Hartree-Focki põhiolek $|\Psi_0\rangle$ kuju

$$|\Psi_0\rangle = |00...01...11\rangle, \tag{1.46}$$

kus N kvantbitti on olekus 1 ning $2K-N$ kvantbitti on olekus 0, ja N on elektronide arv [14]. Jordan-Wigneri teisenduse abil on UCCSD seisund avaldatav kujul

$$|UCCSD\rangle = e^{-iH}|\Psi_0\rangle, \tag{1.47}$$

kus hamiltoniaan H koosneb liikmetest

$$H = \sum_{i}^{N} g_i H_i, \tag{1.48}$$

milles g_i on reaalne koefitsient ja iga liige on produkt

$$H_i = \bigotimes_{k=1}^{n} \sigma_k^i, \tag{1.49}$$

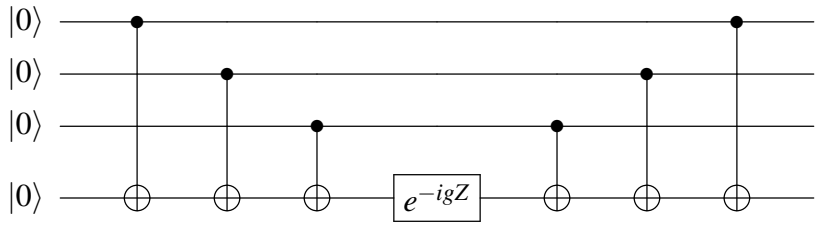
kus σ_k^i on ühikmaatriks I või Pauli maatriks X, Y või Z, mis mõjub kvantbitile k. Lähtudes Trotteri valemist

$$\lim_{n\to\infty}(e^{iA/n}e^{iB/n})^n = e^{i(A+B)} \tag{1.50}$$

saame valemis (1.47) oleva eksponendi lähendada kujul

$$e^{-iH} \approx e^{-ig_1H_1}e^{-ig_2H_2}\dots e^{-ig_nH_n}. \tag{1.51}$$

Avaldises (1.51) olevaid eksponentliikmeid saab esitada kvantahelana. Näiteks hamiltoniaanile $H = Z_1 \otimes Z_2 \otimes Z_3$ vastava eksponentliikme saab esitada ahelana, mis on esitatud joonisel 1.2.



Joonis 1.2: Kvantahel, mis simuleerib hamiltoniaani $Z_1 \otimes Z_2 \otimes Z_3$.

Värav e^{-igZ} teostab kvantbitil g radiaanise pöörde ümber z-telje. Asendades koefitsiendid g_i parameetritega θ_i, saame kvantahelasse sisestada parameetrid, mille minimiseerimist hakkame VQE algoritmis läbi viima. Juhul kui hamiltoniaan sisaldab X või Y väravat, tuleb vastavale kvantbitile teostada baasivahetus

$$X = HZH, \quad Y = SHZHS^\dagger. \tag{1.52}$$

Enne ja pärast joonisel 1.2 välja toodud ahelat mõjub vastavale kvantbitile X värava puhul Hadamardi värav. Y värava puhul enne ahelat mõjub faasi ja Hadamardi värav ning pärast

Hadamardi ja faasi kaasvärav [7].

1.5 Mõõtmine

Viimases alapeatükis viiakse läbi elektronstruktuuri hamiltoniaani mõõtmine. See hamiltoniaan on VQE algoritmi valem (1.9) hamiltoniaan. Selle hamiltoniaani keskväärtuse minimiseerimine hakkab toimuma algoritmis. Esimeses alapeatükis valemis (1.2) välja toodud elektronstruktuuri hamiltoniaani on võimalik lihtsustada kasutades Born-Oppenheimeri lähendust. Kuna molekulis on tuumad elektronidest raskemad, liiguvad need tunduvalt aeglasemalt. Born-Oppenheimeri lähendus seisneb selles, et molekuli tuumad loetakse statsionaarseteks punktideks ja elektronid liiguvad tuumade väljas. Süsteemi hamiltoniaan saab kuju

$$H = -\sum_{i=1}^{N} \frac{1}{2}\nabla_i^2 - \sum_{i=1}^{N}\sum_{A=1}^{M} \frac{Z_A}{r_{iA}} + \sum_{i=1}^{N}\sum_{j>i}^{N} \frac{1}{r_{ij}}, \tag{1.53}$$

kus alles jäävad elektronide kineetilise energia, elektronide ja tuumade ning elektronide omavahelise vastastikmõju liikmed [12]. Molekulaarse süsteemi hamiltoniaan Born-Oppenheimeri lähenduses saab teises kvantiseerimises kuju

$$H = \sum_{p,q} h_{pq} a_p^\dagger a_q + \frac{1}{2}\sum_{p,q,r,s} h_{pqrs} a_p^\dagger a_q^\dagger a_r a_s, \tag{1.54}$$

kus koefitsiendid on defineeritud järgmiselt

$$h_{pq} = \int d\boldsymbol{x} \psi_p^*(\boldsymbol{x}) \left(-\frac{\nabla^2}{2} - \sum_I \frac{Z_i}{|\boldsymbol{r} - \boldsymbol{R}_I|} \right) \psi_q(\boldsymbol{x}) \tag{1.55}$$

$$h_{pqrs} = \int d\boldsymbol{x}_1 d\boldsymbol{x}_2 \frac{\psi_p^*(\boldsymbol{x}_1)\psi_q^*(\boldsymbol{x}_2)\psi_r^*(\boldsymbol{x}_2)\psi_s^*(\boldsymbol{x}_1)}{|\boldsymbol{r}_1 - \boldsymbol{r}_2|}. \tag{1.56}$$

Antud hamiltoniaanile saab rakendada neljandas alapeatükis välja toodud Jordan-Wigneri teisenduse. Jordan-Wigneri teisenduse tulemusena saab hamiltoniaan kuju

$$H = \sum_i f_i \bigotimes_{k=1}^{n} \sigma_k^i, \tag{1.57}$$

kus σ_k^i on ühikmaatriks või Pauli maatriks ja f_i on koefitsient, mis sõltub molekuli geomeetriast [15]. VQE algoritmi jaoks leitakse antud hamiltoniaani keskväärtus. Kuna hamiltonaan on mitme liikme summa, siis on võimalik leida iga liikme keskväärtus eraldi ja neid summeerides leida kogu hamiltoniaani keskväärtus. Z operaatorite keskväärtuste jaoks lihtsalt mõõdetakse vastavad kvantbitid kvantahela lõpus ning arvutatakse keskväärtus. Y ja X operaatorite puhul

teostatakse enne mõõtmist baasiteisendus vastavalt valemile (1.52). Kuna käesolevas töös ei kasutatud kvantarvuti statistilist simulaatorit, vaid olekuvektori simulaatorit, ei ole baasiteisendust vaja teostada, sest keskväärtuse saab simulaatoriga leitud olekuvektoriga otseselt arvutada.

2 Metoodika

2.1 VQE realiseerimine

VQE algoritmi rakendamine teostati programmeerimiskeeles Python (versioon 3.7). Python valiti tingituna selle laialdasest kasutusest teadusarvutustes ning kvantarvutamise pakettidest. Peamised paketid, mida kasutati olid kvantkeemia arvutuspakett `psi4` [16] ja Google kvantarvutite paketid `cirq` [17], `openfermion` [18] ja `qsim` [19]. Lisaks kasutati teaduspakette `scipy` ja `numpy` ning üldiseid pakette `datetime`, `logging`, `multiprocessing`, `sys` ja `time`.

Programm koosneb kahest failist: `vqe_functions.py` ja `run_vqe.py`. Fail `vqe_functions.py` sisaldab kaheksat funktsiooni, mida on vaja VQE algoritmi rakendamiseks. Fail `run_vqe.py` on programmi peafail, mille kasutaja käivitab. Programmi käivitamisel on võimalik anda käsureal ette, kas soovitakse leida energia miinimumväärtus või mitu eri geomeetria energiat. Lisaks on võimalik määrata, mis molekuliga arvutused läbi viiakse.

Programmi käivitamisel esmalt loetakse sisse kasutaja poolt antud parameetrid ning luuakse logifail. Järgnevalt lahkneb programmi töö kaheks sõltuvalt sellest, kas leitakse energia miinimum tasakaaluoleku geomeetriaga või energia väärtused erinevatel aatomituumade vahelistel kaugustel. Esiteks arvutatakse paketti `psi4` kasutades molekuli CCSD operaator. Järgnevalt teisendatakse CCSD operaator kvantbittoperaatorite kujule ning luuakse sellele vastav kvantahel, mis sisaldab tundmatuid parameetreid. Järmise sammuna kasutatakse `psi4`-ga leitud molekulaarset hamiltoniaani ning teisendatakse see ka kvantbittoperaatorite kujule. Nüüd järgneb VQE algoritmi rakendus. Minimiseeritakse keskväärtuse leidmise funktsiooni, mis tagastab kvantahelale, hamiltoniaanile ning parameetrite järjendile vastava keskväärtuse. Minimiseerimine toimub seni, kuni leitakse parameetrite kogum, millele vastab väikseim keskväärtus. Viimase sammuna salvestatakse arvutustulemused.

Mitme energiaväärtuse leidmise protsess on sarnane. Esiteks teostatakse `psi4` arvutused kõigile erinevatele tuumade vahekaugustele. Seejärel paralleelselt mitmel tuumal viiakse sarnaselt miinimumi leidmisele läbi arvutused 15 erineval vahekaugusel. Kuigi teoreetiliselt oleks saanud rohkem punkte korraga arvutada, kuna arvutil oli 64 tuuma, ei olnud see võimalik. Probleemi

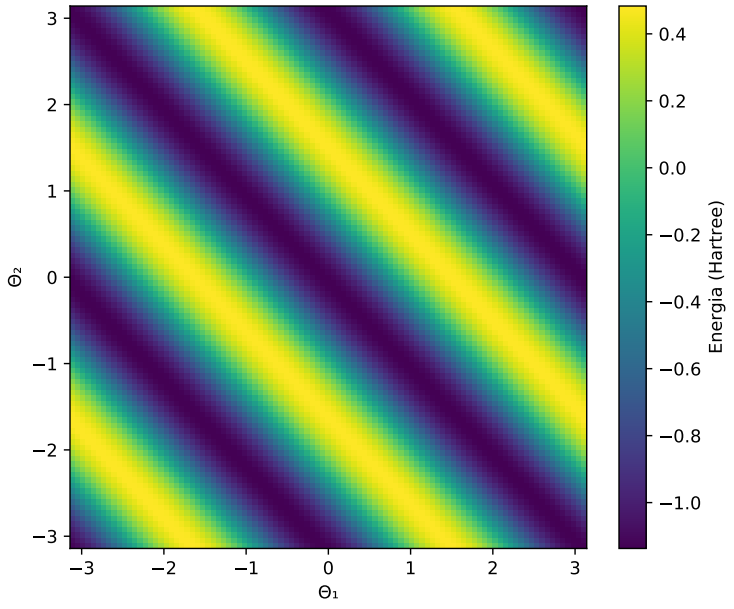
valmistas kvantahela simulaator `qsim`, mis kasutas mitut lõime korraga. Kui üritada läbi viia liiga suur arv arvutusi paralleelselt, ei suuda arvuti protsessor kõiki simulaatori alamprotsesse korraga jooksutada.

2.2 Arvutuste läbiviimine

VQE algoritmi rakendati H_2, LiH ja BeH_2 molekulide energiate arvutamiseks. Igale molekulile leiti miinimumenergia tasakaaluolekule vastava geomeetriaga. Lisaks arvutati igale molekulile miinimumenergiad erinevatel aatomituumade vahekaugustel. Vahekaugusi arvutati 30 punktis vahemikus $0.2 - 3.0\ \mathring{A}$. BeH_2 molekuliga käsitleti ainult juhtu, kus sideme pikkused olid võrdsed. Molekulaarsete orbitaalide leidmiseks võeti aatomorbitaalide baasiks STO-3G. Arvutused viidi läbi süsteemil, mille keskprotsessoriks oli 64 tuumaline AMD (Advanced Micro Devices) Ryzen Threadripper 3970X. Süsteemil oli 256 GB muutmälu.

VQE algoritmi parameetrite optimiseerimiseks kasutati Nelder-Mead'i meetodit [20]. Optimiseerimise täpsuseks võeti 10^{-4}, mis on täpsem kui töös otsitav keemiline täpsus $1.6 \cdot 10^{-3}$ Hartree (Ha). Kõigi parameetrite algväärtuseks valiti null. Parameetrite optimiseerimise vahemikuks valiti $[-\pi, \pi]$. Antud vahemik valiti, kuna parameetrid on kvantvärava R_Z argumendid, mis muutuvad vahemikus $-\pi$ kuni π.

Joonisel 2.1 on välja toodud H_2 molekuli keskväärtuse sõltuvus kahest optimiseerimise parameetrist. Suurematel molekulidel kasvab parameetrite arv koos molekulaarorbitaalide arvuga ning seetõttu pole võimalik sõltuvust joonisel esitada.



Joonis 2.1: Hamiltoniaani keskväärtuse $\langle H \rangle$ sõltuvus kvantahela parameetritest (θ_1, θ_2) molekulaarse vesiniku H_2 näitel.

3 Tulemused ja arutelu

Tänapäeva kvantkeemias on välja arendatud suur hulk meetodeid molekulide põhienergiate leidmiseks. Kõige täpsemaks on antud töös varasemalt käsitletud FCI. Sõltuvalt baasi valikust on võimalik väga täpseid energiaväärtusi leida klassikalisel arvutil. Suur probleem antud lähenemisega on see, et elektronide arvu suurenedes kasvab eksponentsiaalselt FCI lainefunktsiooni komponentide arv ning suureneb arvutuseks vaja minev arvutusjõudlus. Teiste meetodite seas on FCI kõige täpsem, kuid suure arvutusressursi nõudluse tõttu suurtel molekulidel teostamatu.

Suuremate molekulide arvutuste jaoks kasutatakse vähem täpseid, aga tunduvalt vähem arvutusjõudlust nõudvaid meetodeid. Sellised meetodid kaotavad FCI-le omase täpsuse, aga see eest võimaldavad uurida tunduvalt suuremaid molekule. Üheks nendest on näiteks Hartree-Focki teooria.

Kvantarvutite tulevikueesmärk on võimaldada läbi viia FCI täpsusele lähenevaid arvutusi suurematel molekulidel. Aeg, millal kvantarvutid suudavad viia läbi arvutusi, mida klassikalised ei suuda, võib veel olla mitme aasta taga. See on tingitud sellest, et kvantarvutid on mürarikkad ning vajavad veaparanduskoodide kasutamist. Veaparanduskoodid seisnevad selles, et ühe kvantbiti informatsioon salvestatakse mitmesse kvantbitti. Seeläbi säilib arvutusteks vaja minev informatsioon arvutuste käigus. Veaparanduskoodide kasutamise negatiivne külg on see, et kvantahelad, mis neid kasutades, tekivad on nii pikad, et neid pole võimalik lähituleviku kvantarvutitel rakendada kvantväravate vigade tõttu. Lisaks suurendab veaparanduskoodide kasutamine vajaminevat kvantbittide arvu miljonitesse kvantbittidesse.

Siinkohal tuleb sisse VQE algoritm, mis ei kasuta üldjuhul veaparanduskoodi ning on seetõttu piisavalt lühikeste kvantahelatega, et neid rakendada lähituleviku kvantarvutitel [14]. Antud töös oli eesmägiks rakendada VQE algoritm väikeste molekulide simuleerimiseks minimaalses STO-3G baasis.

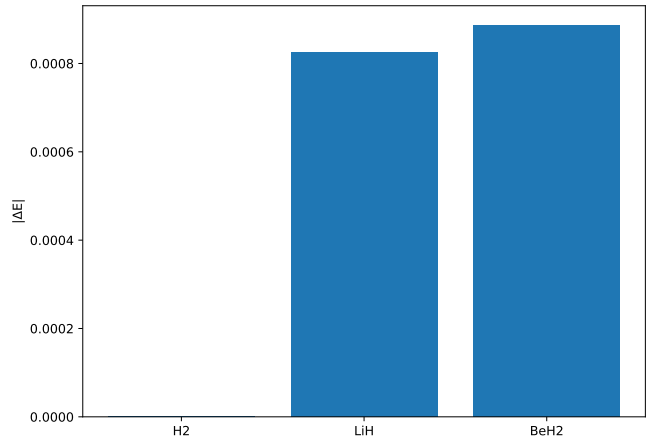
3.1 Tasakaalugeomeetria

Molekulide tasakaalugeomeetria miinimumenergia arvutuste tulemused on välja toodud tabelis 3.1.

Molekul	Energia (Ha)
H_2	-1.137
LiH	-7.882
BeH_2	-15.595

Tabel 3.1: Uuritud molekulide põhioleku energia tasakaalugeomeetrial.

VQE algoritmiga saadud tulemuste võrdluseks arvutati `psi4` programmiga FCI energiad STO-3G baasis. Kõige paremaks võib lugeda vesiniku molekuli tulemust, kuna see erineb FCI tulemusest vaid $2.6 \cdot 10^{-7}$ Ha võrra, mis on tunduvalt väiksem keemilisest täpsusest. Miinimumenergiate erinevused on toodud joonisel 3.1.

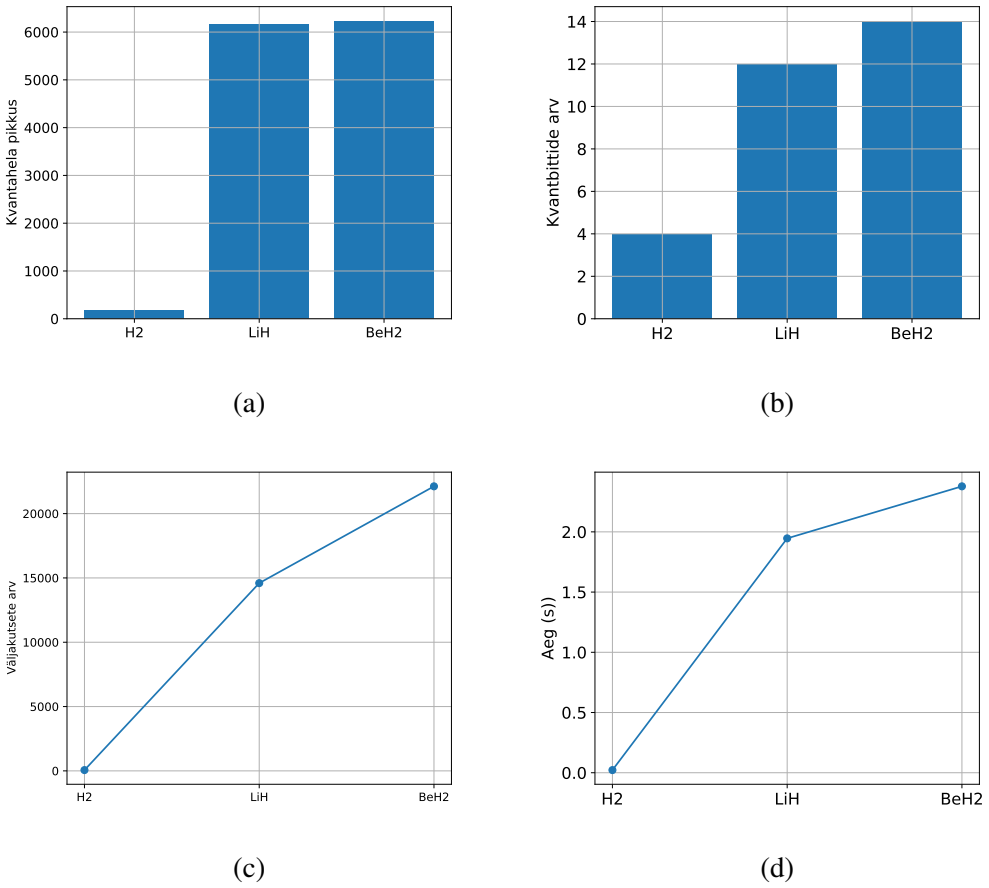


Joonis 3.1: H_2, LiH ja BeH_2 molekulide energiate võrdlus tasakaalugeomeetrial. $|\Delta E|$ näitab FCI ja VQE-ga arvutatud energiate erinevust STO-3G baasis.

On näha, et kõik tasakaaluoleku miinimumenergiate erinevused on väiksemad kui $1.6 \cdot 10^{-3}$ Ha, mida kvantkeemias kasutatakse täpsuse standardina. Tasakaaluolekute arvutused aitavad ka võrrelda VQE algoritmi rakendamist erineva suurusega molekulidel.

Joonisel 3.2 on välja toodud võrdlus VQE algoritmi rakendamisest kolmel molekulil. Nagu on oodata, näeb jooniselt, et suurema elektronide arvuga molekulid vajavad rohkem ressursse

arvutuste läbiviimiseks. Kõik alajoonised on seotud elektronide arvuga molekulides. Kui molekulides on rohkem elektrone, moodustub elektronide aatomorbitaalidest rohkem molekulaarorbitaale, mis omakorda suurendab UCCSD kvantahela pikkust ja kvantahela parameetrite arvu. Optimiseerimise kiirus ning funktsiooni väljakutsete arv sõltub parameetrite arvust, mida on vaja optimiseerida. Kvantbittide arv sõltub molekuli spinnorbitaalide arvust.



Joonis 3.2: (a) Kvantahela pikkus kirjeldab, mitme kvantvärava pikkune kvantahel tuleb koostada antud molekuliga arvutamiseks. (b) Kvantbittide arv kirjeldab, mitut kvantbitti on tarvis antud molekuli VQE algoritmi realiseerimiseks. (c) Optimiseerimisefunktsiooni väljakutsete arv näitab, mitu korda optimisaator leidis kvantahela keskväärtust. (d) Üksiku kvantahela keskväärtuse leidmise aeg kirjeldab, kui palju kulus aega molekuli kvantahela üheks simulatsiooniks.

Kvantarvutuste läbiviimisel ei kasutatud antud töös meetodeid, mis vähendaksid UCCSD operaatorite arvu. Sellest on tingitud kvantahela suur pikkus, mille tõttu ei oleks antud töös loodud kvantahelaid võimalik reaalsetel kvantarvutitel kasutada. Kvantarvutitel tekib iga kvantvärava rakendamisega väike viga kvantbiti olekusse, mis pikkade ahelate korral kaotab ära enamiku kasulikust informatsioonist, mida oleks vaja kvantahela lõpus mõõta. Kui viia läbi UCCSD optimiseerimine, oleks võimalik vähendada kvantahela pikkust, mis viiks kvantahela

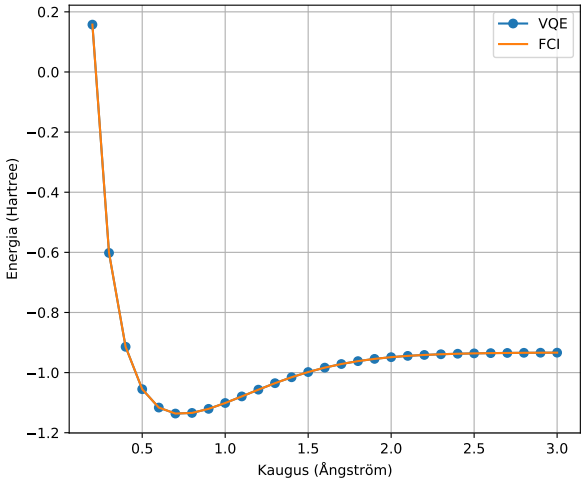
lähemale vormile, mida oleks võimalik reaalsel kvantarvutil rakendada.

Olemasolevate kvantarvutite kvantbittide arv on piisav, selleks et rakendada töös koostatud kvantahelaid. Kvantbittide arvu järgi maailma suurim kvantarvuti on 76 kvantbitiga Jiuzhang nimeline kvantarvuti Hiinas [5]. Lisaks mitmed Google-i [4] ja IBM-i (International Business Machines Corporation) [21] kvantarvutid on sarnaste kvantbittide arvuga. Seetõttu on võimalik 14 kvantbitist kvantahelat rakendada. Samas kvantbittide vähendamist on vaja teostada, sest reaalsel kvantarvutil on kvantahel parem ehitada väiksema arvu kvantbittidega kui kvantarvuti võimaldab, sest siis on võimalik kvaliteetsemad kvantbitid valida ahela rakendamiseks.

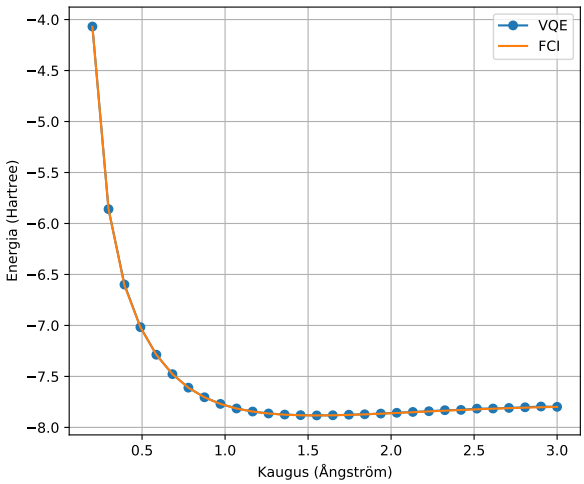
Joonis 3.2 toob välja teise probleemi, mida oleks võimalik optimiseerimisega leevendada. LiH ja BeH_2 molekulidega teeb Nelder-Mead'i optimiseerija liiga suure arvu väljakutseid, mis omakorda suurendab koguarvutuse ajakulu. Arvutusajad olid vastavalt umbes 8 ja 16 tundi. Vesiniku molekulil oli tänu vähestele parameetritele arvutusaeg 3 sekundit. Jällegi, kvantahela optimiseerimisega oleks võimalik operaatorite arvu vähendada, mis omakorda vähendaks parameetrite arvu, mida on vaja optimiseerida. Kvantahela `qsim` olekuvektori simulaatoriga simuleerimimiseks kulus suurematel molekulidel umbes 2 sekundit. Vesiniku arvutused kestsid umbes 0.02 s, mis on tingitud selle lihtsusest.

3.2 Energia sõltuvus geomeetriast

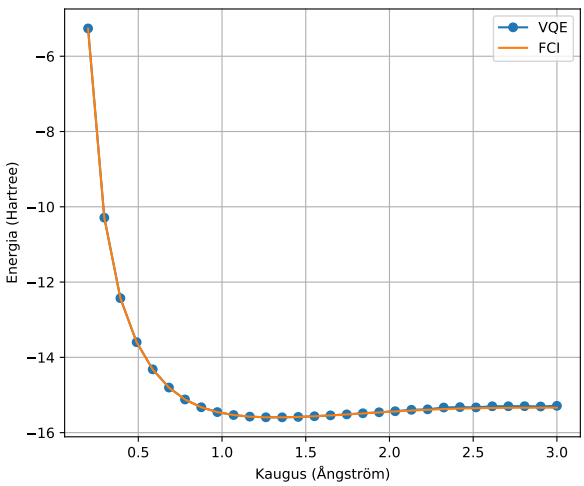
VQE algoritmiga leitud energiatele leiti sarnaselt miinimumidele FCI võrdlusenergiad. Energiate leidmiseks viidi läbi arvutused 30 punktis, tuumade kaugustel vahemikus $0.2 - 3.0\,\mathring{A}$. Kõigi kolme molekuli tulemused on välja toodud joonisel 3.3.



(a)



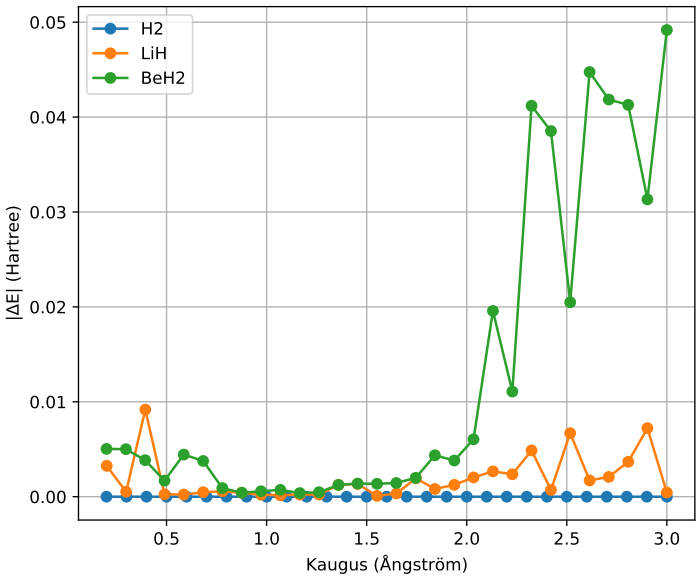
(b)



(c)

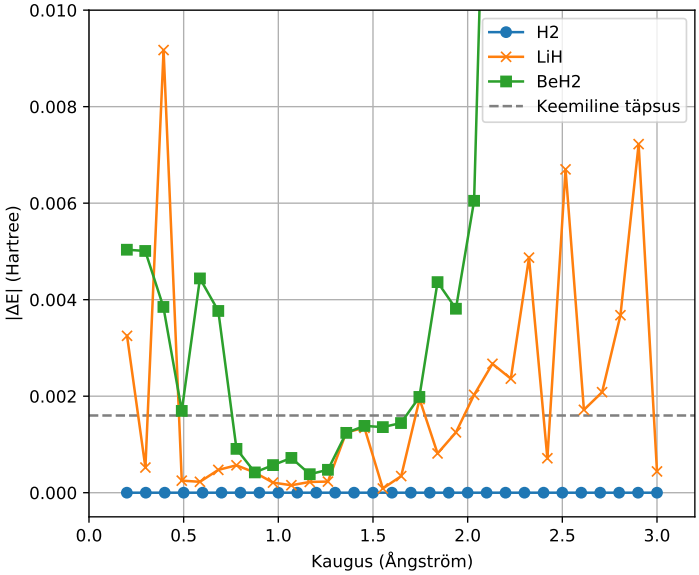
Joonis 3.3: VQE (sinine punkt) ja FCI energiad (oranz joon) erinevatel tuumade vahekaugustel (a) H_2, (b) LiH ja (c) BeH_2 molekulide jaoks.

Joonise 3.3 järgi on VQE saanud suhteliselt hea täpsusega tulemused võrreldes FCI-ga. Täpsema võrdluse jaoks VQE algoritmi ja FCI meetodi energiate vahel on joonisel 3.4 välja toodud kolme molekuli VQE ja FCI energiate vahede absoluutväärtused. Jooniselt on näha, et BeH_2 puhul kasvab VQE viga suureks kaugustel, mis on suuremad kui 2 $\AA$.



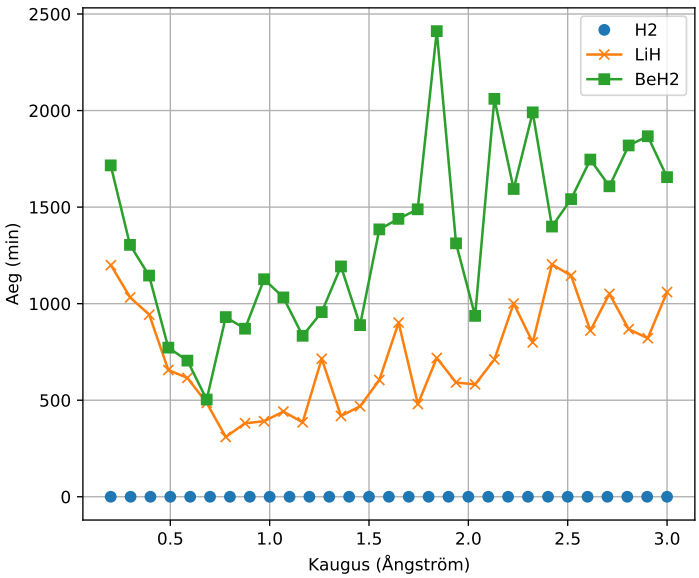
Joonis 3.4: VQE algoritmiga ja FCI meetodiga leitud molekulide energiate erinevused tuumade eri vahekaugustel.

Joonisel 3.5 on esitatud joonise 3.4 energia erinevuse suurendus vahemikus $0-0.01$ Ha. Lisatud on ka joon energial $1.6 \cdot 10^{-3}$ Ha ehk 0.043 eV, mis tähistab arvutustes soovitud keemilist täpsust.



Joonis 3.5: VQE algoritmiga ja FCI meetodiga leitud molekulide energiate erinevused erinevatel tuumade kaugustel.

Joonis 3.5 näitab, et H_2 molekuli tulemus on kõigil väärtustel parema täpsusega kui vajalik keemiline täpsus. LiH ja BeH_2 molekulide puhul on näha, et aatomituumade kaugustel, mis on lähedased tasakaaluoleku tuumade kaugustega, on arvutus piisavalt täpne. Kaugustel, kus molekul on ebastabiilsem, kuna tuumade vahekaugus on väga väike või suur, on arvutusviga tunduvalt suurem. See võib olla tingitud sellest, et optimisaatoril on ebastabiilsematel molekuli geomeetriatel raskem leida õiget energia miinimumi. Selle hüpoteesi võib esitada joonise 3.6 alusel, kus on kujutatud arvutusajad erinevatel kaugustel.



Joonis 3.6: VQE algoritmi miinimumi leidmiseks kulunud aeg erinevatel tuumade vahekaugustel kolmel molekulil.

On näha, et tasakaaluoleku kauguste lähedased punktid minimiseeritakse kiiremini ja parema täpsusega. Erandiks on H_2 molekul, mille puhul toimub ühtlaselt kiire minimiseerimine. LiH ja BeH_2 üksiku punkti minimeerimine jäi vahemiku $7-30$ tundi. Siinkohal tuleb ära märkida, et energiaid arvutati paralleelselt. See ei mõjutanud punkti miinimumi leidmiseks kulunud aega, kuna üks miinimumi leidmine koormas ainult ühte arvuti tuuma.

Antud töö VQE algoritmi implementatsioon on tunduvalt aeglasem klassikalistel arvutitel kasutatavatest kvantkeemia arvutusmeetoditest. Kui VQE algoritmil kulus mitmeid tunde ühe energiaväärtuse leidmiseks, suutis klassikalisi meetodeid kasutav `psi4` sama energia leida mõne sekundiga. Selle koha peal tuleks käsitleda mitut argumenti. Esiteks ei käsitletud antud töös ühtegi UCCSD optimismeerimismeetodit, mis lühendaks kvantahelat, vähendaks optimiseerimisparameetrite ja kvantbittide arvu. Teiseks ja veelgi tähtsamaks punktiks on see, et VQE algoritm on mõeldud reaalsete kvantarvutite jaoks. Praegu on kvantarvutid veel tasemel, kus näiteks O'Malley et al [22] ja Google Sycamore [6] VQE kvantarvutused käsitlevad suhteliselt lihtsaid molekule, mida klassikalise arvutiga tunduvalt lihtsam ja kiirem arvutada. Lähitulevikus, kui ehitatakse kvantarvutid, mille kvantbittide arv on tuhandetes [21], tuleb esile

kvantarvutite eelis klassikaliste arvutite ees. Kvantarvutite võime kodeerida suurtes kogustes informatsiooni kvantbittide abil annab suure eelise suurte ja kompleksete molekulide täpseteks arvutusteks.

Kokkuvõte

Antud töö raames realiseeriti VQE algoritm kvantarvuti olekuvektori simulaatoril. Arvutati STO-3G baasis H_2, LiH ja BeH_2 molekulide põhioleku energiad ning nimetatud molekulide energiad eri geomeetriatel. Optimiseerimiseks kasutati Nelder-Mead'i meetodit. Arvutatud energiate vajalikuks täpsuseks võeti $1.6 \cdot 10^{-3}$ Hartree. Leitud energiate võrdluseks arvutati FCI energiad klassikaliste meetoditega.

Põhioleku energiate väärtused saadi kõik keemilise täpsuse piirides. Põhioleku energiate arvutustes kogutud andmeid kasutati ka VQE algoritmi võrdluseks erineva suurusega molekulidel. LiH ja BeH_2 molekulide kvantahelad olid umbes 6000 tuhande kvantvärava pikkused. Kvantahelate optimiseerimist ei käsitletud antud töös. Selle tagajärjena olid ka arvutusajad vastavalt 8 ja 16 tundi. See on tingitud suurest parameetrite arvust kvantahelas, mida oli vaja optimiseerida. H_2 molekuli miinimumenergia leidmiseks kulus 3 sekundit. See oli tingitud antud molekuli lihtsusest.

H_2 energiate arvutuste jaoks oli tulemus keemilise täpsuse piires. LiH ja BeH_2 arvutus ei saavutatud keemilist täpsust. Mõlema molekuli juures täheldati head täpsust tasakaaluoleku lähedal, sellest eemaldudes suurenes viga mitmekordseks. Töö autori arvates on see tingitud sellest, et vähem stabiilsemate molekuli geomeetriate juures on optimisaatoril raskem leida õiget miinimumenergiat.

Töös käsitletud teemasid oleks võimalik edasi arendada rakendades optimiseerimist UCCSD kvantahelale. Lühem kvantahel ning väiksem parameetrite arv vähendaks arvutusaega. Lisaks saab uurida erinevate optimiseerimismeetodite efektiivsust VQE algoritmis. Töös käsitleti ainult ideaalset olekuvektori simulaatorit. Edasiarendusena oleks võimalik rakendada algoritmi statistilisel simulaatoril või isegi võimalusel reaalsel kvantarvutil. See nõuaks lisatööd reaalsetel kvantarvutitel tekkivate vigade vähendamisel.

Kirjandus

[1] P. Benioff, "Quantum mechanical hamiltonian models of turing machines," Journal of Statistical Physics **29**, 515–546 (1982).

[2] R. P. Feynman, "Simulating physics with computers," International Journal of Theoretical Physics **21**, 467–488 (1982).

[3] I. L. Chuang, L. M. K. Vandersypen, X. Zhou, D. W. Leung, ja S. Lloyd, "Experimental realization of a quantum algorithm," Nature **393**, 143–146 (1998).

[4] F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends, R. Biswas, S. Boixo, F. G. S. L. Brandao, D. A. Buell, B. Burkett, Y. Chen, Z. Chen, B. Chiaro, R. Collins, W. Courtney, A. Dunsworth, E. Farhi, B. Foxen, A. Fowler, C. Gidney, M. Giustina, R. Graff, K. Guerin, S. Habegger, M. P. Harrigan, M. J. Hartmann, A. Ho, M. Hoffmann, T. Huang, T. S. Humble, S. V. Isakov, E. Jeffrey, Z. Jiang, D. Kafri, K. Kechedzhi, J. Kelly, P. V. Klimov, S. Knysh, A. Korotkov, F. Kostritsa, D. Landhuis, M. Lindmark, E. Lucero, D. Lyakh, S. Mandrà, J. R. McClean, M. McEwen, A. Megrant, X. Mi, K. Michielsen, M. Mohseni, J. Mutus, O. Naaman, M. Neeley, C. Neill, M. Y. Niu, E. Ostby, A. Petukhov, J. C. Platt, C. Quintana, E. G. Rieffel, P. Roushan, N. C. Rubin, D. Sank, K. J. Satzinger, V. Smelyanskiy, K. J. Sung, M. D. Trevithick, A. Vainsencher, B. Villalonga, T. White, Z. J. Yao, P. Yeh, A. Zalcman, H. Neven, ja J. M. Martinis, "Quantum supremacy using a programmable superconducting processor," Nature **574**, 505–510 (2019).

[5] H.-S. Zhong, H. Wang, Y.-H. Deng, M.-C. Chen, L.-C. Peng, Y.-H. Luo, J. Qin, D. Wu, X. Ding, Y. Hu, P. Hu, X.-Y. Yang, W.-J. Zhang, H. Li, Y. Li, X. Jiang, L. Gan, G. Yang, L. You, Z. Wang, L. Li, N.-L. Liu, C.-Y. Lu, ja J.-W. Pan, "Quantum computational advantage using photons," Science **370**, 1460–1463 (2020).

[6] F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends, S. Boixo, M. Broughton, B. B. Buckley, ja et al., "Hartree-fock on a superconducting qubit quantum computer," Science **369**, 1084–1089 (2020).

[7] M. A. Nielsen ja I. L. Chuang, *Quantum Computation and Quantum Information* (Cambridge University Press, Cambridge, 2016).

[8] P. Kaye, R. Laflamme, ja M. Mosca, *An Introduction to Quantum Computing* (Oxford University Press, Oxford, 2010).

[9] D. J. Griffiths, *Introduction to Quantum Mechanics* (Pearson Education Limited, Harlow, 2014).

[10] J. R. McClean, J. Romero, R. Babbush, ja A. Aspuru-Guzik, "The theory of variational hybrid quantum-classical algorithms," New Journal of Physics **18**, 023023 (2016).

[11] P. Atkins ja L. Jones, *Keemia alused* (W. H. Freeman and Company, New York, 2012).

[12] A. Szabo ja N. Ostlund, *Modern Quantum Chemistry: Introduction to Advanced Electronic Structure theory* (Dover Publications, INC, Mineola, New York, 1996).

[13] T. Helgaker, P. Jørgensen, ja J. Olsen, *Molecular Electronic-Structure Theory* (John Wiley Sons, Ltd, West Sussex, England, 2012).

[14] S. McArdle, S. Endo, A. Aspuru-Guzik, S. C. Benjamin, ja X. Yuan, "Quantum computational chemistry," Reviews of Modern Physics **92** (2020).

[15] Y. Cao, J. Romero, J. P. Olson, M. Degroote, P. D. Johnson, M. Kieferová, I. D. Kivlichan, T. Menke, B. Peropadre, N. P. D. Sawaya, ja et al., "Quantum chemistry in the age of quantum computing," Chemical Reviews **119**, 10856–10915 (2019).

[16] D. G. A. Smith, L. A. Burns, A. C. Simmonett, R. M. Parrish, M. C. Schieber, R. Galvelis, P. Kraus, H. Kruse, R. Di Remigio, A. Alenaizan, A. M. James, S. Lehtola, J. P. Misiewicz, M. Scheurer, R. A. Shaw, J. B. Schriber, Y. Xie, Z. L. Glick, D. A. Sirianni, J.Õ'Brien, J. M. Waldrop, A. Kumar, E. G. Hohenstein, B. P. Pritchard, B. R. Brooks, H. F. Schaefer, A. Y. Sokolov, K. Patkowski, A. E. DePrince, U. Bozkaya, R. A. King, F. A. Evangelista, J. M. Turney, T. D. Crawford, ja C. D. Sherrill, "Psi4 1.4: Open-source software for high-throughput quantum chemistry," The Journal of Chemical Physics **152**, 184108 (2020).

[17] Cirq Developers, "Cirq," (2021). `https://github.com/quantumlib/Cirq`.

[18] J. R. McClean, K. J. Sung, I. D. Kivlichan, Y. Cao, C. Dai, E. Schuyler Fried, C. Gidney, B. Gimby, P. Gokhale, T. Häner, T. Hardikar, V. Havlíček, O. Higgott, C. Huang, J. Izaac, Z. Jiang, X. Liu, S. McArdle, M. Neeley, T. O'Brien, B. O'Gorman, I. Ozfidan, M. D. Radin, J. Romero, N. Rubin, N. P. D. Sawaya, K. Setia, S. Sim, D. Steiger, M. Steudtner, Q. Sun, W. Sun, D. Wang, F. Zhang, ja R. Babbush, "OpenFermion: the electronic structure package for quantum computers," Quantum Science and Technology **5**, 034014 (2020).

[19] Quantum AI team and collaborators, "qsim," (2021). `https://github.com/quantumlib/qsim`.

[20] J. A. Nelder ja R. Mead, “A simplex method for function minimization,” The Computer Journal **7**, 308–313 (1965).

[21] K. Wehden, I. Faro, ja J. Gambetta, “IBM’s roadmap for building an open quantum software ecosystem,” https://www.ibm.com/blogs/research/2021/02/quantum-development-roadmap/ (2021).

[22] P. J. J. O’Malley, R. Babbush, I. D. Kivlichan, J. Romero, J. R. McClean, R. Barends, J. Kelly, P. Roushan, A. Tranter, N. Ding, ja et al., “Scalable quantum simulation of molecular energies,” Physical Review X **6** (2016).

A Lisad

A.1 vqe_functions.py

```
import logging
import time as time

import cirq as cirq
import numpy as numpy
import qsimcirq as qsimcirq
from openfermion import (FermionOperator, MolecularData, bravyi_kitaev,
                         get_fermion_operator, jordan_wigner,
                         uccsd_convert_amplitude_format)
from openfermionpsi4 import run_psi4
from scipy.optimize import minimize, shgo
from sympy import Symbol


def get_qubit_operators(molecular_data):
    """Leiab kvantbittoperaatorid psi4 arvutatud andmetest.

    Args:
        molecular_data (MolecularData): MolecularData, millele on tehtud psi4 arvutused

    Returns:
        list: Kvantbittoperaatorite jarjend
    """

    # uksik ja kaksik ergastuste amplituudid psi4 arvutusest.
    single_amplitudes = molecular_data.ccsd_single_amps
    double_amplitudes = molecular_data.ccsd_double_amps

    if (isinstance(single_amplitudes, numpy.ndarray) or
        isinstance(double_amplitudes, numpy.ndarray)):
        single_amplitudes_list, double_amplitudes_list =
            uccsd_convert_amplitude_format( single_amplitudes, double_amplitudes)

    fermion_operator_list = list()
    # uksik ergastused.
    for (i, j), t_ik in single_amplitudes_list:
        i, j = int(i), int(j)
        ferm_op = FermionOperator(((i, 1), (j, 0)), 1.)
            - FermionOperator(((j, 1), (i, 0)), 1.)
        fermion_operator_list.append(ferm_op)

```

```
    # Kaksik ergastused.
    for (i, j, k, l), t_ijkl in double_amplitudes_list:
        i, j, k, l = int(i), int(j), int(k), int(l)
        ferm_op = (FermionOperator(((i, 1), (j, 0), (k, 1), (l, 0)), 1.)
            - FermionOperator(((l, 1), (k, 0), (j, 1), (i, 0)), 1.))
        fermion_operator_list.append(ferm_op)

    # Jordan-Wigneri teisendus.
    qubit_operator_list = list()
    for fermion_operator in fermion_operator_list:
        qubit_operator_list.append(jordan_wigner(fermion_operator))

    return qubit_operator_list


def create_uccsd(qubit_operator_list, qubit_count, param):
    """Loob kvantbittoperaatorite jarjendist UCCSD kvantahela.

    Args:
        qubit_operator_list (list[QubitOperator]): Kvantbittoperaatorite jarjend
        qubit_count (int): Kvantbittide arv
        param (String)): Optimiseerimis parameetri tahistus

    Returns:
        Circuit: UCCSD kvantahel
    """

    circuit = cirq.Circuit()
    # Erinevate kvantbittoperaatorite loomine.
    for i in range(len(qubit_operator_list)):

        qubits = cirq.LineQubit.range(qubit_count)
        sub_circuit = cirq.Circuit()
        terms_list = qubit_operator_list[i].terms

        # Tundmatu parameeteri loomine.
        param_string = param + "{}".format(i)
        temp_param = Symbol(param_string)

        # Erinevate eksponentide loomine.
        for term in terms_list:

            # Baasi vahetus.
            moment1 = cirq.Moment()
            moment2 = cirq.Moment()

            for basis in term:
                if basis[1] == 'X':
                    q = qubits[int(basis[0])]
                    moment1 = moment1.with_operation(cirq.H(q))

                if basis[1] == 'Y':
                    q = qubits[int(basis[0])]
                    moment1 = moment1.with_operation(cirq.S(q)**(-1))
                    moment2 = moment2.with_operation(cirq.H(q))


            sub_circuit.append(moment1)
            sub_circuit.append(moment2)
```

```
101
102             # Eksponendi kvantahela loomine.
103             exponent = cirq.Circuit()
104             max_qubit = 0
105             for basis in term:
106                 if basis[0] > max_qubit:
107                     max_qubit = basis[0]
108
109             for basis in term:
110                 if basis[0] < max_qubit:
111                     exponent.append(cirq.CNOT(qubits[basis[0]], qubits[max_qubit]),
112                                     strategy = cirq.InsertStrategy.NEW)
113
114             rotate_z = cirq.rz(-2 * terms_list[term].imag *  temp_param)
115
116             exponent_reverse = exponent**(-1)
117             exponent.append([rotate_z(qubits[max_qubit]), exponent_reverse],
118                             strategy = cirq.InsertStrategy.NEW)
119
120             sub_circuit.append(exponent)
121
122             # Baasi vahetus tagasi.
123             moment3 = cirq.Moment()
124             moment4 = cirq.Moment()
125
126             for basis in term:
127                 if basis[1] == 'X':
128                     q = qubits[int(basis[0])]
129                     moment3 = moment3.with_operation(cirq.H(q))
130
131                 if basis[1] == 'Y':
132                     q = qubits[int(basis[0])]
133                     moment3 = moment3.with_operation(cirq.H(q))
134                     moment4 = moment4.with_operation(cirq.S(q))
135
136
137             sub_circuit.append(moment3)
138             sub_circuit.append(moment4)
139
140         circuit.append(sub_circuit, strategy = cirq.InsertStrategy.NEW)
141
142     return circuit
143
144
145 def initial_hartree_fock(electron_count, qubit_count):
146     """Loob Hartree-Focki pohi oleku |11..100...0>.
147
148     Args:
149         electron_count (int): Elektronide arv molekulis
150         qubit_count (int): Kvantbittide arv kvantahelas
151
152     Returns:
153         Circuit: Hartree-Focki pohiseisund UCCSD ahela algusesse
154     """
155
156     circuit = cirq.Circuit()
157     qubits = cirq.LineQubit.range(qubit_count)
158
159     i = 0
```

```
    while i < electron_count:
        circuit.append(cirq.X(qubits[i]), strategy = cirq.InsertStrategy.INLINE)
        i += 1

    return circuit


def get_measurement_hamiltonian(molecular_data):
    """Loob mootmis hamiltoniaani.

    Args:
        molecular_data (MolecularData): MolecularData koos psi4 arvutustega

    Returns:
        QubitOperator: Mootmis hamiltoniaan
    """

    one_body_integrals = molecular_data.one_body_integrals
    two_body_integrals = molecular_data.two_body_integrals
    orbitals = molecular_data.canonical_orbitals
    molecular_data.save()
    molecule_qubit_hamiltonian = jordan_wigner(get_fermion_operator(
                            molecular_data.get_molecular_hamiltonian()))

    return molecule_qubit_hamiltonian


def get_measurement_pauli_sum(molecule_qubit_hamiltonian, qubit_count):
    """Loob Pauli summa mootmis hamiltoniaanist.

    Args:
        molecule_qubit_hamiltonian (QubitOperator): Mootmis hamiltoniaan
        qubit_count (int): Kvantbittide arv

    Returns:
        PauliSum: Pauli summa
    """
    pauli_sum = None
    qubits = cirq.LineQubit.range(qubit_count)
    terms = molecule_qubit_hamiltonian.terms

    # Erinevad hamiltoniaani osad.
    for term in terms:
        pauli_string = None

        # Identiteedi operaatori erand.
        if len(term) == 0:
            if pauli_sum is  None:
                pauli_sum = cirq.I(qubits[0]) * terms[term].real
            else:
                pauli_sum += cirq.I(qubits) * terms[term].real

            continue

        # Kvantbittoperaatorite lisamine Pauli summasse.
        for basis in term:
            if basis[1] == 'X':
                # Kui pauli_string-i pole lisatud varavat.
                if pauli_string is None:
```

```
                    pauli_string = cirq.X(qubits[basis[0]])
                # Kui varav on juba olemas.
                else:
                    pauli_string = pauli_string * cirq.X(qubits[basis[0]])

            elif basis[1] == 'Y':
                if pauli_string is None:
                    pauli_string = cirq.Y(qubits[basis[0]])
                else:
                    pauli_string = pauli_string * cirq.Y(qubits[basis[0]])

            elif basis[1] == 'Z':
                if pauli_string is None:
                    pauli_string = cirq.Z(qubits[basis[0]])
                else:
                    pauli_string = pauli_string * cirq.Z(qubits[basis[0]])

        # Koik kokku liita uheks Pauli summaks.
        if pauli_sum is not None:
            pauli_sum += pauli_string * terms[term].real
        else:
            pauli_sum =  pauli_string * terms[term].real

    return pauli_sum


def get_qubit_map(qubit_count):
    """Loob expectation_from_state_vector() funktsiooni jaoks qubit_map-i.
    kvantbitt[j] -> arv j.

    Args:
        qubit_count (int): Kvantbittide arv

    Returns:
        [dict]: {LineQubit: int} Qubitmap
    """
    qubit_map = dict()
    qubits = cirq.LineQubit.range(qubit_count)
    i = 0

    for qubit in qubits:
        qubit_map[qubit] = i
        i += 1

    return qubit_map


def get_expectation_value(x, *args):
    """Kasutada koos scipy.minimize optimiseerijaga.
    Leiab ahelale, hamiltoniaanile ja parameetritele vastava keskvaartuse.

    Args:
        x (list): Parameetrite jarjend
        *args (): simulator, UCCSD, pauli_sum, qubit_map

    Returns:
        float: Keskvaartus
    """
    start_time = time.time()
```

```
278
279     assert len(args) == 4
280     simulator, uccsd, pauli_sum, qubit_map = args
281
282     # Parameetrite lisamine kvantahelale.
283     resolver_dict = dict()
284     for k in range(len(x)):
285         resolver_dict.update({'t{}'.format(k): x[k]})
286     resolver = cirq.ParamResolver(resolver_dict)
287
288     # Qsim simuleerib kvantahela.
289     result = simulator.simulate(uccsd, resolver)
290     state_vector = result.final_state_vector
291
292     norm = numpy.linalg.norm(state_vector)
293     state_vector = state_vector / norm
294
295     # Keskvaartuse leiab kvantahela olekuvektori ja Pauli summaga.
296     expectation_value = pauli_sum.expectation_from_state_vector(state_vector, qubit_map)
297
298     elapsed_time = time.time() - start_time
299     # Aja logimine kui on vaja.
300
301     return expectation_value.real
302
303
304 def single_point_calculation(values):
305     """Arvutab uhe punkti miinimum energia vaartuse
306
307     Args:
308         values (List): [MolecularData, faili_nimi]
309
310     Returns:
311         List: Leitud vaartused
312     """
313     molecular_data = values[0]
314     file_name = values[1]
315     length = molecular_data.description
316     logging.info("Starting expectation value calculations for length %s.", length)
317
318     electron_count = molecular_data.n_electrons
319     qubit_count = molecular_data.n_qubits
320
321     # Kvantahela loomine.
322     qubit_operator_list = get_qubit_operators(molecular_data)
323     uccsd = initial_hartree_fock(electron_count, qubit_count)
324     unitary = create_uccsd(qubit_operator_list, qubit_count, 't')
325     uccsd.append(unitary, strategy = cirq.InsertStrategy.NEW)
326
327     # Mootmis hamiltoniaani loomine.
328     hamiltonian = get_measurement_hamiltonian(molecular_data)
329     qubit_map = get_qubit_map(qubit_count)
330     pauli_sum = get_measurement_pauli_sum(hamiltonian, qubit_count)
331
332     # Simulaatori seaded.
333     options = {'t': 6}
334     simulator = qsimcirq.QSimSimulator(options)
335     cirq.DropEmptyMoments().optimize_circuit(circuit = uccsd)
336
```

```
337     start_time = time.time()
338
339     # Miinimumi leidmine.
340     optimize_result = minimize(get_expectation_value,
341                         x0 = numpy.zeros(len(qubit_operator_list)),
342                         method = 'Nelder-Mead',
343                         args = (simulator, uccsd, pauli_sum, qubit_map),
344                         options = {'disp' : True, 'ftol': 1e-4,
345                         'maxiter': 100000, 'maxfev': 100000})
346
347     elapsed_time = time.time() - start_time
348     logging.info(optimize_result)
349     logging.info("Elapsed time: %s", elapsed_time)
350
351     energy_min = optimize_result.fun
352     nfev = optimize_result.nfev
353     nit = optimize_result.nit
354
355     # Tulemuste salvestamine faili.
356     file = open(file_name, "a")
357     file.write("{}, {}, {}, {}, {}, \n".format(energy_min, nfev,
358                                             nit, elapsed_time,
359                                             length))
360     file.close()
361     logging.info("Result at %s saved.", length)
362
363     return energy_min, nfev, nit, elapsed_time
```

A.2 run_vqe.py

```
import datetime as datetime
import logging
import multiprocessing as mp
import sys
import time as time

import cirq as cirq
import numpy as numpy
import pandas as pandas
import qsimcirq as qsimcirq
from openfermion import MolecularData
from openfermionpsi4 import run_psi4
from scipy.optimize import minimize

from vqe_functions import *


def calculate_minimum(molecule_name, basis, multiplicity, charge):
    """ Leiab miinimum energia molekuli tasakaalu olekule.

    Args:
        molecule_name (String): Molekuli keemiline luhend
        basis (int): Arvutuste baas
        multiplicity (int): Molekuli multiplicity
        charge (int): Molekuli laeng
    """
    logging.info("Calculating %s minimum.", molecule_name)

    # Molekulide tasakaalu olekute geomeetriad.
    min_geometry_dict = {'H2': [[ 'H', [ 0, 0, 0]],
                            [ 'H', [ 0, 0, 0.74]]],
                    'LiH': [['Li', [0, 0, 0]] ,
                            ['H', [0, 0, 1.5949]]],
                    'BeH2': [['Be', [ 0, 0, 0 ]],
                            ['H', [ 0, 0, 1.3264]],
                            ['H', [ 0, 0, -1.3264]]],
                    'H2O': [['O', [-0.053670056908, -0.039737675589, 0]],
                            ['H', [-0.028413670411,  0.928922556351, 0]],
                            ['H', [0.880196420813,  -0.298256807934, 0]]]}
    geometry = min_geometry_dict[molecule_name]

    molecular_data = MolecularData(geometry, basis, multiplicity,
        charge, filename = './data/{}_min_molecule.data'.format(molecule_name))

    #Psi4 arvutused.
    molecular_data = run_psi4(molecular_data,
                            run_scf=True,
                            run_mp2=True,
                            run_cisd=True,
                            run_ccsd=True,
                            run_fci=True)

    # Miinimum vaartuse leidmine.
    file_name = "./results/VQE_min_{}_{}.csv".format(molecule_name,
                                                datetime.datetime.now())
```

```
    min_result = single_point_calculation([molecular_data, file_name])

    # Molekuli info logimine.
    logging.info("Electron count: %s", molecular_data.n_electrons)
    logging.info("Qubit count: %s", molecular_data.n_qubits)
    logging.info("Orbital count: %s", molecular_data.n_orbitals)
    logging.info("Results saved.")


def calculate_scan(molecule_name, basis, multiplicity, charge, counts):
    """ Leiab miinimum energia molekuli eri tuumade vahelistel kaugustel

    Args:
        molecule_name (String): Molekuli keemiline luhend
        basis (int): Arvutuste baas
        multiplicity (int): Molekuli multiplicity
        charge (int): Molekuli laeng
        counts (int): Arvutus punktide arv
    """

    length_bounds = [0.2, 3]
    logging.info("Calculating %s scan.", molecule_name)
    file_name = "./results/VQE_scan_{}_{}.csv".format(molecule_name,
                                            datetime.datetime.now())
    molecular_data_list = list()

    # Erinevate kauguste psi4 arvutuste labi viimine enne VQE algoritmi rakendamist.
    for length in numpy.linspace(length_bounds[0], length_bounds[1], counts):
        length = round(length, 3)
        while True:
            geometry_dict = {'H2': [[ 'H', [ 0, 0, 0]],
                                    [ 'H', [ 0, 0, length]]],
                             'LiH': [['Li', [0, 0, 0]] ,
                                    ['H', [0, 0,  length]]],
                             'BeH2': [['Be', [ 0, 0, 0 ]],
                                    ['H', [ 0, 0,  length]],
                                    ['H', [ 0, 0, - length]]],
                             'H2O': [['O', [0, 0, 0]],
                                    ['H', [numpy.sin(0.9163) * length,
                                        numpy.cos(0.9163) * length, 0]],
                                    ['H', [- numpy.sin(0.9163) * length,
                                        numpy.cos(0.9163) * length, 0]]]}

            geometry = geometry_dict[molecule_name]

            try:
                logging.info("Trying psi4 calculation at length %s.", length)
                molecular_data = MolecularData(geometry, basis, multiplicity,
                    charge, filename = './data/{}_{}_scan_molecule.data'.format(
                                    molecule_name, length), description=str(length))

                molecular_data = run_psi4(molecular_data,
                                          run_scf=True,
                                          run_mp2=True,
                                          run_cisd=True,
                                          run_ccsd=True,
                                          run_fci=True)

                logging.info("Psi4 calculations were succesful.")
```

```
115
116                 molecular_data_list.append([molecular_data, file_name])
117                 break
118             except Exception as exc:
119                 # Kui antud molekuli geomeetriaga ei suutnud psi4 vajalike arvutusi teha,
120                 # suurendatakse tuumade vahelist vahekaugust 0.001 ja proovitakse uuesti.
121                 logging.error(exc)
122                 length += 0.001
123                 logging.info("New length set: %s", length)
124
125     # Erinevate kauguste miinimumid arvutatakse paraleelselt.
126     pool = mp.Pool(processes= 10)
127     result = pool.map(single_point_calculation, molecular_data_list)
128
129
130 def main():
131     """
132     -min molecule \n
133     -scan counts molecule \n
134     Molecules: H2, LiH, BeH2
135     """
136     start = time.time()
137     # Programmi argumentide sisse lugemine.
138     args = sys.argv[1:]
139     assert len(args) >= 2
140     min_mode = ('-min' in args)
141     scan_mode = ('-scan' in args)
142     molecule_name = args[-1]
143     if scan_mode:
144         counts = int(args[-2])
145
146     # Logimis faili ules seadmine.
147     logging.basicConfig(
148     level=logging.DEBUG,
149     format="%(asctime)s [%(levelname)s] %(message)s",
150     handlers=[
151         logging.FileHandler("./logs/LOG_{}_{}.log".format(molecule_name,
152                                                           datetime.datetime.now())),
153         logging.StreamHandler()
154     ])
155
156     # Koigi arvutuste jaoks kehtivad jargmised tingimused.
157     basis = 'sto-3g'
158     multiplicity = 1
159     charge = 0
160
161     # Viiakse labi kas miinimum arvutus voi mitme punkti arvutus.
162     if min_mode:
163         calculate_minimum(molecule_name, basis, multiplicity, charge)
164     elif scan_mode:
165         calculate_scan(molecule_name, basis, multiplicity, charge, counts)
166     else:
167         print("No mode selected!")
168         exit()
169
170     elapsed_time = time.time() - start
171     logging.info("Total programm run time: %s s.", elapsed_time)
172     exit()
173
```

```
174
175 # Promgramm algab:
176 if __name__ == "__main__":
177     main()
```

Lihtlitsents lõputöö reprodutseerimiseks ja üldsusele kättesaadavaks tegemiseks

Mina, Marko Raidlo

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) minu loodud teose

Molekulaarse süsteemi põhioleku energia arvutamine omaväärtusülesande variatsioonilise kvantlahendamise algoritmi abil,

mille juhendajadeks on Veiko Palge ja Dirk Oliver Theis, reprodutseerimiseks eesmärgiga seda säilitada, sealhulgas lisada digitaalarhiivi DSpace kuni autoriõiguse kehtivuse lõppemiseni.

2. Annan Tartu Ülikoolile loa teha punktis 1 nimetatud teos üldsusele kättesaadavaks Tartu Ülikooli veebikeskkonna, sealhulgas digitaalarhiivi DSpace kaudu Creative Commonsi litsentsiga CC BY NC ND 3.0, mis lubab autorile viidates teost reprodutseerida, levitada ja üldsusele suunata ning keelab luua tuletatud teost ja kasutada teost ärieesmärgil, kuni autoriõiguse kehtivuse lõppemiseni.

3. Olen teadlik, et punktides 1 ja 2 nimetatud õigused jäävad alles ka autorile.

4. Kinnitan, et lihtlitsentsi andmisega ei riku ma teiste isikute intellektuaalomandi ega isikuandmete kaitse õigusaktidest tulenevaid õigusi.

Marko Raidlo

26.05.2021