FYS4150 - Computational Physics

PROSJEKT 2

Å løse Schrödingers likning for ett og to elektroner

Jon Vegard Sparre
jonvsp@uio.no

Anne-Marthe Hovda
annemmho@uio.no

Dato: 4. oktober 2015

Sammendrag

I dette prosjektet har vi løst Schrödingerlikninga for harmonisk oscillator med ett og to elektroner. Vi har brukt Jacobis metode og egenverdi-løseren til Armadillo-biblioteket for å løse den numerisk. De numeriske resultatene har blitt sammenliknet med de analytiske resultatene til M. Taut[2]. Vi har kun sett på løsningene for relativ avstand med og uten Coulomb-frastøtning. I tillegg har vi også sett på presisjon og stabilitet.

Lenke til Jon Vegards GitHub-domene: https://github.com/jonvegards/FYS4150

Introduksjon

Problemet som vi kommer til å løse i dette prosjektet er Schrödingerlikninga (SL) for en 3D harmonisk oscillator med ett og to elektroner. Vi har et sfærisk symmetrisk potensiale, som gjør at vi trenger kun å løse den radielle SL, *i.e.* vi reduserer det tredimensjonale problemet til et éndimensjonalt et. Den radielle delen av SL er da, for en harmonisk oscillator

$$-\frac{\hbar^2}{2m}\left(\frac{1}{r^2}\frac{\mathrm{d}}{\mathrm{d}r}r^2\frac{\mathrm{d}}{\mathrm{d}r}-\frac{l(l+1)}{r^2}\right)R(r)+\frac{m\omega^2r^2}{2}R(r)=ER(r).$$

Det er essensielt denne likninga vi skal løse gjennom hele prosjeket i forskjellige former. Metoden vi skal bruke her, ble vi kjent med i prosjekt 1. Vi tar utgangspunkt i trepunktsformelen for en annenderivert, diskretiserer funksjonen i n steg med en steglengde h, setter det opp som en tridiagonal matrise og løser matriselikninga med ei egna algoritme.

I løpet av prosjektet vil vi utvikle vår egen numeriske løsning ved hjelp av *Jacobis algoritme*. I tillegg til dette vil vi også bruke Armadillo til å løse problemet. Begge metodene vil så bli sammenliknet med den analytiske løsningen til Taut [2]. Vi vil også bruke MATLAB for å lage plott (og litt egenverdikontroll underveis).

Teori

SL er gitt i sfæriske koordinater, s.a. $r \in [0, \infty)$. Denne likninga inneholder noen stygge annenderiverte som vi vil endre på, så vi substituerer $R(r) = (1/r)u(r)$ og får,

$$-\frac{\hbar^2}{2m}\frac{\mathrm{d}^2}{\mathrm{d}r^2}u(r)+\left(\frac{\hbar^2}{2m}\frac{l(l+1)}{r^2}+\frac{m\omega^2r^2}{2}\right)u(r)=Eu(r).$$

Randbetingelsene her er greie å finne ut av, siden vi ikke vil at $R(r)$ skal divergere så må $u(0) = 0$, det er lett å se ut i fra hvoran vi har definert $R(r)$. Potensialet vårt øker med r^2, dette tilsier at sannsynligheten vil avta jo lengre unna vi er fra origo og dermed blir $u(\infty) = 0$.

Prøver vi å putte likninga inn i et program slik den er nå, så vil vi få noen ubehagelige overraskelser hva gjelder avrundingsfeil, overflow og andre ting vi ikke vil ha i en numerisk løsning. Vi innfører derfor dimensjonsløse variable, da blir resultatet automagisk på den lengdeskalaen problemet krever at det er. Først setter vi $\rho = (1/\alpha)r$, α er en konstant med dimensjon lengde. Ved å sette inn dette får vi

$$-\frac{\hbar^2}{2m\alpha^2}\frac{\mathrm{d}^2}{\mathrm{d}\rho^2}u(\rho) + \left(\frac{\hbar^2}{2m\alpha^2}\frac{l(l+1)}{r^2} + \frac{m\omega^2\alpha^2\rho^2}{2}\right)u(\rho) = Eu(\rho),$$

vi vil ha den annenderiverte for seg selv, så vi ganger med $2m\alpha^2/\hbar^2$ på begge sider og får

$$-\frac{\mathrm{d}^2}{\mathrm{d}\rho^2}u(\rho) + \left(\frac{l(l+1)}{r^2} + \frac{m^2\omega^2\alpha^4\rho^2}{\hbar^2}\right)u(\rho) = \frac{2m\alpha^2}{\hbar^2}Eu(\rho),$$

en siste endring vi gjør er å sette $m^2\omega^2\alpha^4/\hbar^2 = 1$ og å introdusere $\lambda = 2m\alpha^2 E/\hbar^2$. Vi skal i dette prosjektet kun se på løsninger med $l = 0$, alt dette gir oss da likninga vi skal implementere i programmet,

$$-\frac{\mathrm{d}^2}{\mathrm{d}\rho^2}u(\rho) + \rho^2 u(\rho) = \lambda u(\rho).$$

Den analytiske løsninga for dette problemet gir oss egenverdiene $\lambda_1 = 3$, $\lambda_2 = 7$ og $\lambda_3 = 11$.

Det er ingen grunn til å vente med å se på likninga for problemet med to elektroner, så vi gyver løs på den med en gang. Vi vil så gå gjennom hvordan vi diskretiserer likninga og får den på matriseform, slik vi gjorde i prosjekt 1. For et system med to elektroner vil SL (uten Coulombinteraksjoner) være

$$\left(-\frac{\hbar^2}{2m}\frac{\mathrm{d}^2}{\mathrm{d}r_1^2} - \frac{\hbar^2}{2m}\frac{\mathrm{d}^2}{\mathrm{d}r_2^2} + \frac{1}{2}k_1 r_1^2 + \frac{1}{2}k_2 r_2^2\right)u(r_1, r_2) = E^{(2)}u(r_1, r_2),$$

hvor r_1 og r_2 hendholdsvis er posisjonene til elektron 1 og elektron 2, energien $E^{(2)}$ er den samla energien til systemet og $k_i = m\omega_i^2$. Vi introduserer så den relative koordinaten $r = r_1 - r_2$ og massesenterkoordinaten $R = (1/2)(r_1 + r_2)$, SL tar da formen

$$\left(-\frac{\hbar^2}{m}\frac{\mathrm{d}^2}{\mathrm{d}r^2} - \frac{\hbar^2}{4m}\frac{\mathrm{d}^2}{\mathrm{d}R^2} + \frac{1}{4}m\omega_r^2 r^2 + m\omega_R^2 R^2\right)u(r, R) = E^{(2)}u(r, R).$$

Dette er heldigvis en separabel differensiallikning, så vi kan sette $u(r, R) = \psi(r)\phi(R)$ og bruke at energien er gitt som $E^{(2)} = E_r + E_R$. Her er vi bare interessert i å løse for $\psi(r)$. Vi legger også til Coulombinteraksjonen, som avhenger av avstanden mellom elektronene,

$$V(r) = \frac{\beta e^2}{r}.$$

Vi får da

$$\left(-\frac{\hbar^2}{m}\frac{\mathrm{d}^2}{\mathrm{d}r^{\,2}} + \frac{1}{4}m\omega_r^2 r^2 + \frac{\beta e^2}{r}\right)\psi(r) = E_r\psi(r).$$

Vi gjentar substitusjonsprosessen som vi gjorde over, $r = (1/\alpha)$. Vi setter $\alpha = \hbar/m\beta e^2$, $\omega_r^2 = (1/4)(mk\alpha^4/\hbar^2)$ og definerer $\lambda = m\alpha^2 E/\hbar$, dette gir oss

$$-\frac{\mathrm{d}^2}{\mathrm{d}\rho^2}\psi(\rho) + \left(\omega_r^2\rho^2 + \frac{1}{\rho}\right)\psi(\rho) = \lambda\psi(\rho).$$

Vi ser at eneste forskjellen fra i stad er at vi har fått et ekstra potensialledd, hvilket betyr at vi kan lage én algoritme som løser for begge tilfellene.

Metode

Nå som vi har stablet på beina likninga vi skal løse, så kan vi ta en titt på den numeriske delen. Vi skal bruke trepunktsformelen for en annenderivert, som vi her kan skrive som

$$u'' = \frac{u(\rho + h) - 2u(\rho) + u(\rho - h)}{h^2} + \mathcal{O}(h^2),$$

hvor h er steglengden vår. For å diskretisere likninga så kan vi definere

$$h = \frac{\rho_{\max} - \rho_{\min}}{n_{\text{steg}}},$$

dette gir oss $\rho_i = \rho_{\min} + ih$, for $i = 0, 1, \ldots, n_{\text{steg}}$. Vi kan så skrive SL på formen

$$-\frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} + V_i u_i = \lambda u_i,$$

hvor $u_{i+1} = u(\rho_i + h)$ etc. og $V_i = \omega_r^2\rho_i^2 + 1/\rho_i$. Dette skriver vi som en tridiagonal matrise der diagonalelementene defineres til å

være $d_i = (2/h^2) + V_i$ og elementene på diagonalene over og under hoveddiagonalen til $e_i = -1/h^2$. På matriseform blir dette

$$\begin{pmatrix} d_1 & e_1 & 0 & \dots & \dots & 0 \\ e_1 & d_2 & e_2 & 0 & \dots & 0 \\ 0 & \ddots & \ddots & \ddots & \vdots & e_{n_{\text{steg}}-1} \\ \vdots & \ddots & \ddots & \ddots & e_{n_{\text{steg}}-1} & d_{n_{\text{steg}}-1} \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ \vdots \\ u_{n_{\text{steg}}} \end{pmatrix} = \lambda \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ \vdots \\ u_{n_{\text{steg}}} \end{pmatrix}.$$

Dette likner veldig på den matriselikninga vi løste i prosjekt 1, men forskjellen nå er at vi har en egenverdilikning. Dette innebærer at når vi løser systemet så er det egenvektorene (opphøyd i andre, slik vi pleier å gjøre med bølgefunksjoner) vi skal plotte mot avstandsvektorene vi lager. For ett-elektrontilfellet vil vi få et plott som viser hvor det er mest sannsynlig å finne elektronet, mens for to-elektrontilfellet vil vi få et plott som viser den mest sannsynlige avstanden mellom elektronene.

Vi kan nå se på algoritmen vi skal bruke for å løse dette systemet. Den kalles *Jacobis algoritme* og går ut på at man skal rotere matrisa A til den er diagonal. En egenverdilikning med ei diagonalmatrise må nødvendigvis ha egenverdiene må diagonalen, så grunnen til å løse den på denne måten her er ganske åpenbar. Algoritmen er beskrevet i større detalj i kompendiet til Morten Hjort-Jensen [1, p. 215-220], så her vil vi bare gjengi hovedpoenga i den. Vi utfører med rotasjonen med ei antisymmetrisk rotasjonsmatrise S,

$$A' = S^T A S.$$

hvor S er

$$S = \begin{pmatrix} 1 & 0 & \dots & \dots & \dots & \dots & \dots & 0 \\ 0 & 1 & 0 & \dots & \dots & \dots & \dots & \vdots \\ \vdots & 0 & 1 & 0 & \dots & \dots & \dots & 0 \\ \vdots & \dots & 0 & \cos\theta & 0 & \dots & 0 & \sin\theta \\ \vdots & \vdots & \dots & 0 & 1 & \dots & \dots & 0 \\ \vdots & \vdots & \dots & \dots & \dots & \dots & 0 & \vdots \\ \vdots & \vdots & \dots & 0 & \dots & 0 & 1 & 0 \\ 0 & 0 & \dots & -\sin\theta & 0 & \dots & 0 & \cos\theta \end{pmatrix}.$$

Denne matrisa gjør en rotasjon i et n-dimensjonalt Euklidsk rom. På komponentform har vi

$$S_{kk} = S_{ll} = \cos\theta, \quad s_{kl} = -s_{lk} = \sin\theta, \quad s_{ii} = 1, \quad i \neq k, i \neq l.$$

Siden vi vil at alle elementer som ikke er på diagonalen til den roterte A, skal bli null, så må vi finne en vinkel θ som gjør at det blir slik. Vi velger oss derfor det største elementet på ikke-diagonalen i A og setter θ slik at dette elementet blir null, ved å alltid velge det største elementet, sørger vi for at for hver iterasjon, gjør matrisa litt mer diagonal. Vi kan da finne resten av elementene i den nye matrisa A' ut i fra den θ vi finner. Ved å se på (2×2)-tilfellet av matrisa A' så får vi bare ett ikke-diagonalt element å sette lik null (husk at matrisa er symmetrisk), og det er

$$A'_{kl} = A_{kl}(c^2 - s^2) + (Akk - A_{ll})cs = 0,$$

gjør vi litt om på denne likninga, og bruker at $\tan\theta = t = s/c$, så får vi

$$1 - t^2 - t\frac{A_{ll} - A_{kk}}{A_{kl}} = 0 \Rightarrow t^2 + 2\tau t - 1 = 0.$$

Vi definerte i overgangen her $A_{ll} - A_{kk}/(2A_{kl}) = \tau$. Denne andregradslikninga må vi løse for å finne vinkelen θ vi skal rotere matrisa A med og samtidig sette det største elementet i A' til null. Når vi løser denne andregradslikninga så får vi to løsninger. Hvilken skal vi velge? Jo, vi velger den som lager minst bry for oss, og det er den minste vinkelen. Jo mindre vi roterer matrisa for hver gang, jo mindre vil andre elementer i A som i utgangspunktet var null, bli forskjøvet fra null. På denne måten gjør vi det så effektivt som mulig. Dette er i grunn hele essensen i algoritmen: finne største element i A, regne ut θ, rotere, gjenta til A er diagonal.

Videre vil vi se litt på hvordan vi kan utføre matrisemultiplikasjonen på en lur måte. Matrisa S vil bare endre på elementer i rader og kolonner k, l og l, k i A, dette kan vi bruke når vi skal implementere algoritmen i programmet vårt. Vi vil da skrive ei `for`-løkke som går over alle indeksverdier i, og for hvert element som har A_{ik} og A_{il} med $i \neq l, k$, så får vi ei endring. Vi gjør altså følgende utregninger for hver gang vi skal gjøre en rotasjon,

$$\begin{aligned} A'_{ik} &= cA_{ik} - sA_{il} \\ A'_{ki} &= A'_{ik} \\ A'_{il} &= cA_{il} + sA_{ik} \\ A'_{li} &= A'_{il}, \end{aligned}$$

hvor $i \neq k, l$, $c = \cos\theta$ og $s = \sin\theta$. Vi har her utnytta at den nye matrisa også vil bli symmetrisk. For elementene på diagonalen i A

får vi

$$\begin{aligned} A'_{kk} &= c^2 A_{kk} - 2csA_{kl} + s^2 A_{ll} \\ A'_{ll} &= s^2 A_{kk} + 2csA_{kl} + c^2 A_{ll} \\ A'_{kl} &= 0 \\ A'_{lk} &= 0, \end{aligned}$$

Hvor vi i de to siste linjene hardkoder at elementene skal være null siden det er disse elementene vi tar utgangspunkt i for å finne rotasjonsvinkelen θ. Dette er så å si alt vi trenger for å finne egenverdiene vi er på jakt etter. Nå vil vi se på hvordan vi finner egenvektorene. Disse vil nemlig endre seg for hver rotasjon. Siden vi starter med egenverdilikninga $AR = \vec{\lambda}R$, hvor R inneholder alle egenvektorene som kolonnevektorer og $\vec{\lambda}$ er en vektor med de tilhørende egenverdiene, så ser vi at etter endt rotasjonsalgoritme så må vi ha $A_\lambda \mathbb{1}^{n\times n} = \vec{\lambda}\mathbb{1}^{n\times n}$, hvor matrisa A_λ har egenverdiene på diagonalen. Vi kan altså gjøre en motsatt rotasjon på $\mathbb{1}^{n\times n}$ for å finne egenvektorene samtidig som vi roterer $A \to A_\lambda$. Denne rotasjonen skjer akkurat på samme måte som for A,

$$\begin{aligned} R'_{ik} &= cR_{ik} - sR_{il} \\ R'_{il} &= cR_{il} + sR_{ik}, \end{aligned}$$

hvor vi looper over $i \neq k, l$. Nå har vi alt vi trenger for å kunne løse egenverdilikninga vi er gitt!

Før vi ser på resultatene, så skal vi se på hvordan programmet fungerer i praksis. Vi har skrevet programmet slik at vi definerer alle variable og parametre for ett-elektron- og to-elektrontilfellet hver for seg og kaller så på funksjone `JacobiRotation` som igjen kaller på de andre funksjonene for å gjøre utregningene. Ved å bygge opp programmet på denne måten blir det enklere å gjøre enhetstester på det.

Gangen i programmet er som følger: Vi definerer alle vektorer, matriser og variable som er nødvendig for utregningene, *i.e.* A, $\vec{\rho}$ og steglengde h. Vi setter n til å være dimensjonene til matrisene, slik at $\vec{\rho}$ må være $n + 2$ lang for å få med endepunktene. Endepunktene vet vi at skal være null, så de er ikke med i matrisa A. Når vi kaller på funksjonen `JacobiRotation` så sender vi inn n, A, $\vec{w}$ og R, henholdvis matrisedimensjon, matrisa selv, vektor til å inneholde indeksverdier og identitetsmatrisa R. De tre sistnevnte blir endret på globalt av funksjonen slik at vi får hentet ut verdiene vi skal ha når vi skal printe ut resultatene. Før første rotasjon kaller vi på funksjonen `FindingMaximumElementOnNonDiagonal`, den finner største

elementet i matrisa A ved å loope over den øvre triangulære delen av matrisa (minner om at $A^T = A$). Funksjonen gir oss indeksene k og l som forteller oss hvilket element vi skal sette til null og hvor vi skal plassere $\sin\theta$ og $\cos\theta$ i rotasjonsmatrisa S. `JacobiRotation` går så inn i ei `while`-løkke som kaller tre funksjoner for å gjennomføre en iterasjon. Denne løkka går så lenge det største matriseelementet i A er større enn en toleranse ϵ vi definerer. Det første `while`-løkka gjør er å sette $c = s = 0$ for å forsikre oss om at det ikke blir noe kluss i rotasjonen, deretter kaller den på funksjonen `findSinCos` som finner verdiene til $\sin\theta$ og $\cos\theta$ som skal brukes når vi foretar rotasjonen. I denne funksjonen er det en `if`-test som sørger for at vi velger den minste rotasjonen som er mulig. Neste linje i løkka vår kaller på `rotateAmatrix` som foretar selve rotasjonen slik vi har skildra tidligere. Etter å ha rotert så må vi finne det største elementet i den roterte matrisa og så sjekker testen i `while`-løkka om matrisa er blitt diagonal nok før den går videre til en ny iterasjon.

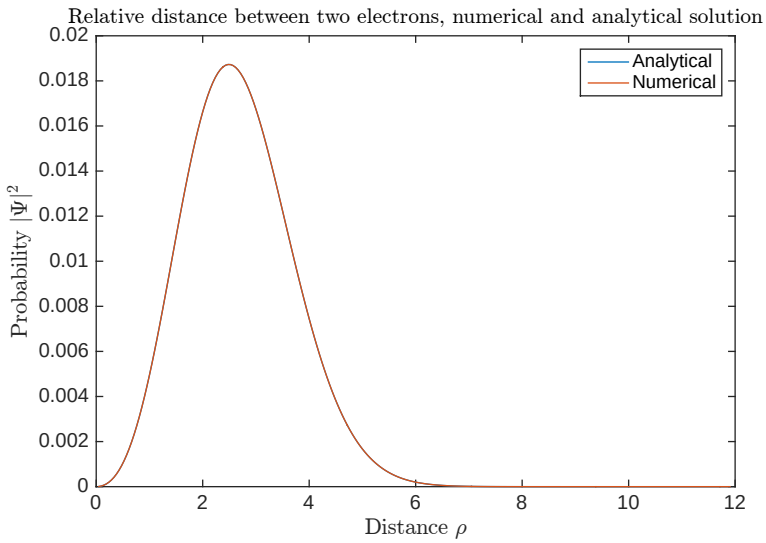
Som allerede nevnt så må vi også finne egenvektorene til egenverdiene vi regner ut. Dette har vi løst ved å bruke Armadillos sorteringsfunksjon. Ved å lage en `uvec`-vektor `w` til å ta vare på indeksverdier så kan vi skrive `w=sort_index(A.diag());`. Dette vil gi oss en vektor `w` som inneholder som første element indeksen til den kolonna i `A` som inneholder den laveste egenverdien, andre element, viser til nest laveste egenverdi og så videre. Disse indeksverdiene bruker vi så på matrisa `R` når vi skal ha tak i de tilhørende egenvektorene. Så når vi skriver `R.col(w(0))` så får vi ut en kolonnevektor som er egenvektoren til den laveste egenverdien.

For å se til at programmet vårt fungerer som det skal, så har vi lagt inn enhetstester. Disse testene tester programmet for kjente tilfeller, slik at vi vet hva som bør være resultatet. Vi har funksjoner som tester `FindMaximumElementOnNonDiagonal` og `JacobiRotation` på to måter. Førstnevnte funksjon får tilsendt ei matrise hvor vi veit hva det største elementet er, og så printer testfunksjonen `MaxElementTest` ut hva den finner. Sistnevnte blir først testa av `TwoByTwoMatrixTest` med ei 2×2-matrise for å sjekke om det holder med én iterasjon for å få ei diagonal matrise, resultatet blir printa ut så man kan se om den fungerer. `JacobiRotation` blir også testa på tilfellet harmonisk oscillator med ett elektron, vi kjenner egenverdiene til å være $\lambda_0 = 3$, $\lambda_1 = 7$ og $\lambda_2 = 11$, testfunksjonen `EigenvalueTest` printer da ut egenverdiene `JacobiRotation` finner.

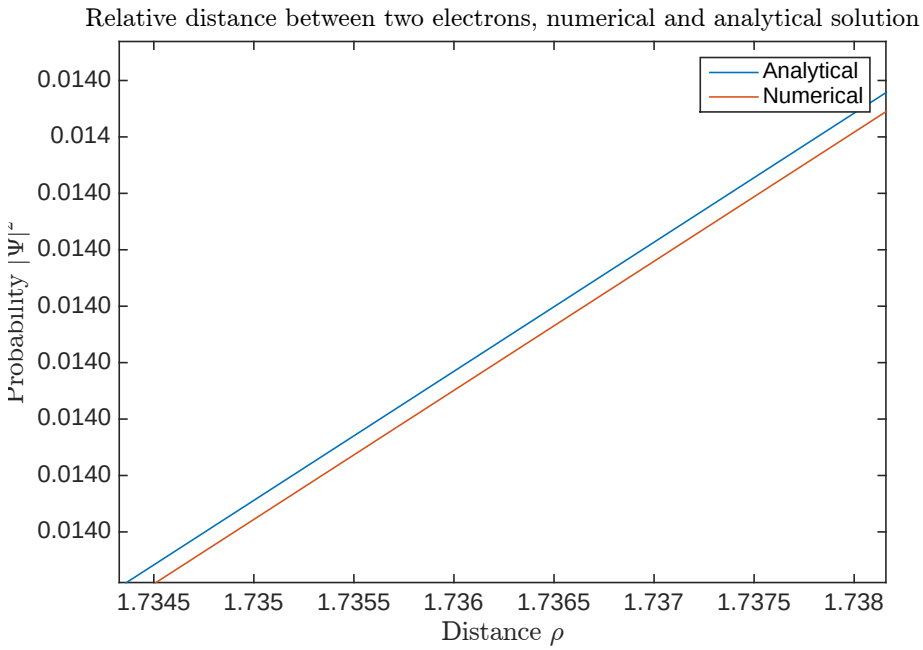
Resultater

Våre resultater kan vises ved hjelp av noen av plottene vi plottet i MATLAB.

Figur 1 viser sannsynlighetsfordelingen til den relative distansen mellom to elektroner, sannsynligheten $|\Psi|^2$ plottet mot den relative distansen ρ. Dette plottet viser oss hvor stor relativ distanse det er størst sannsynlighet for at det er mellom de to elektronene. Vi har her plottet både den numeriske og den analytiske løsningen fra Taut [2], slik at vi kan sjekke de opp mot hverandre. Ut i fra plottet i figur 1 ser vi at den numeriske løsningen stemmer godt overens med den analytiske. Figur 2 viser det samme plottet, men her har vi zoomet inn for sjekke hvor lik den numeriske løsningen er til den analytiske. Her ser vi at den numerisk løsningen ikke er helt lik den analytiske slik som figur 1 kunne få oss til å tro. Dersom man ser på aksene her så ser man at forskjellen mellom de to løsningene er ganske liten og det er rimelig å si at vår numeriske løsning er en god tilnærming til den analytiske. Her har vi valgt å sette maks-verdien til den relative distansen til $\rho_{max} = 12$. Valget av ρ_{max} vil bli diskutert mer i detalj, se tabell 3 for en oversikt over hva som skjer ved forskjellige verdier for ρ_{max}.

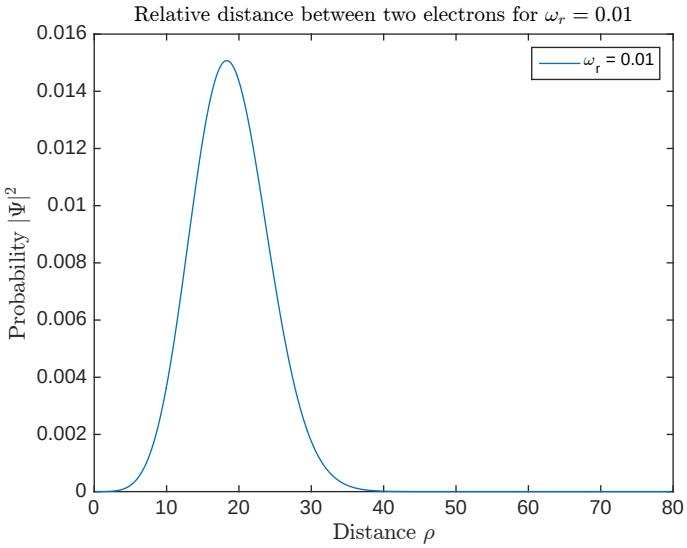


Figur 1: *Den relative distansen mellom to elektroner, numerisk og analytisk løsning. Vi ser at den numeriske løsningen legger seg fint oppå den analytiske.*

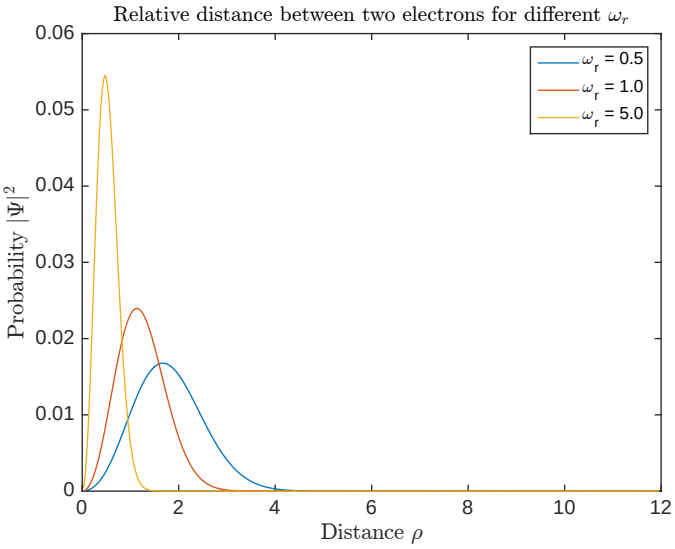


Figur 2: *Den relative distansen mellom to elektroner, når vi har zoomet inn.*

I figur 3 har vi et plott som viser sannsynlighetsfordelinga til den relative distansen mellom to elektroner med en fast verdi for ω_r, hvor ω_r er en faktor som justerer styrken til oscillator-potensialet. Her har vi valgt en høy maks-verdi for distansen, $\rho_{maks} = 80$, dette er fordi det var kun ved en slik høy verdi at vi fikk et stabilt resultat, henviser igjen til videre diskusjon av valget senere i rapporten og tab. 3. Videre viser figur 4 sannsynlighetsfordelinga til den relative distansen for flere forskjellig verdier for ω_r, dette illustrerer hvordan forskjellig styrke på potensialet vil påvirke distansen mellom elektronene. Her ser vi tydelig at ved høyere verdier av ω_r er det høyere sannsynlighet for at elektronene er nærmere hverandre enn ved en lavere verdi av ω_r, dette gir mening siden ω_r er her et mål på styrken til potensialet, og ved et sterkere potensiale vil elektronene *føle* en større kraft og bli dytta nærmere hverandre. Styrken på potensialet vil holde elektronene nærmere og vil i større grad motvirke den frastøtende krafta mellom elektronene. Vi ser også at for større verdier av ω_r så er sannsynlighetsfordelinga spissere, dette tilsier at distansen er skarpere definert ved høyere ω_r, det er et mindre intervall for distansen mellom elektronene.



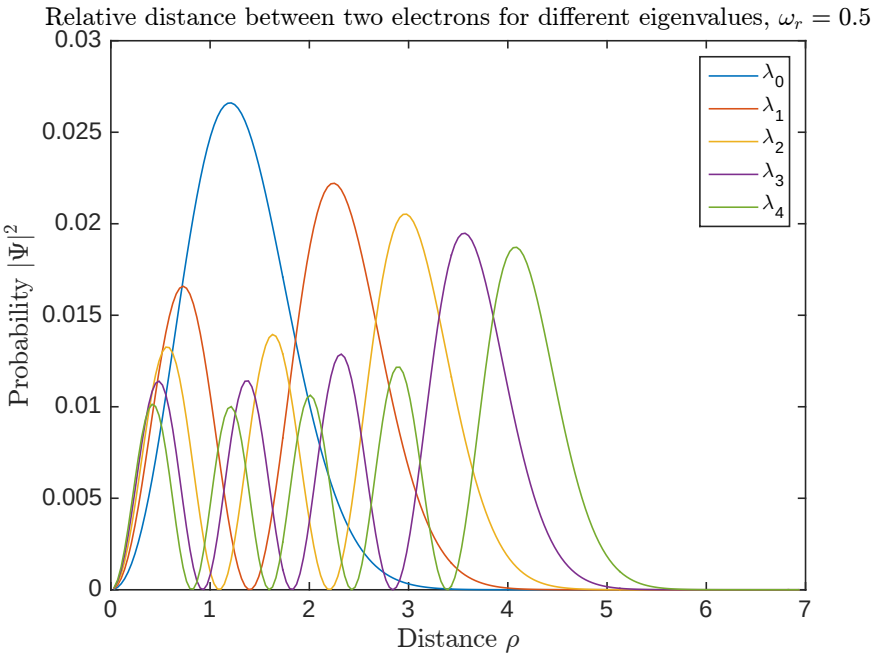
Figur 3: *Den relative distansen mellom to elektroner for* $\omega_r = 0.01$ *og* $n = 400$. *Toppunktet til sannsynlighetsfordelinga ligger cirka ved* $\rho = 20$.



Figur 4: *Den relative distansen mellom to elektroner for forskjellige verdier av* ω_r *ved* $n = 400$. *Toppunktene til sannsynlighetsfordelingene ligger nærmere null enn da* $\omega_r = 0.01$, i.e. *elektronene er nå nærmere hverandre.*

Vi har også lagt ved figur 5. Denne figuren viser hvordan valget

av egenverdiene, λ, påvirker sannsynlighetsfordelingen. Egenverdiene tilsvarer energien til egentilstandene, og ved den laveste egenverdien λ_0 ser vi at det kun er én topp på sannsynlighetsfordelinga, dette gir mening da vi vet at i grunntilstanden er det kun én mulig konfigurasjon av elektronene. I første eksiterte tilstand derimot er det to mulige konfigurasjoner av elektronene og dette gjenspeiles av at sannsynlighetsfordeling har to topper, hvor den ene er litt høyere – det er den tilstanden som det er høyest sannsynlighet for. Videre følger de neste energitilstandene med økende mulige elektronkonfigurasjoner, dette kommer tydelig frem i fig. 5 selv om det er noe rotete med alle de forskjellige sannsynlighetsfordelingene.



Figur 5: *Illustrasjon av egenverdienes påvirkning på sannsynlighetsfordelingen*

Numerisk stabilitet og presisjon

Vi vil her se litt på vårt programs numeriske stabilitet og presisjon. Vi har to hovedparamatre vi kan endre for å forbedre stabiliteten og presisjonen til programmet. Først kan vi se på antall iterasjoner som er nødvendig for at det største elementet i matrisa vi roterer er mindre enn vår $\epsilon = 10^{-8}$. I tab. 1 ser vi resultatene for forskjellige n. Vi ser at programmet vårt kjører med en god del færre iterasjoner

enn teoretisk forventa, noe som kan ha sammenheng med vår valgte ϵ. Velger vi en mindre ϵ må vi ha flere iterasjoner for å få det største elementet i matrisa til å komme nærmere null.

n	Teoretisk	Faktisk
10	300	100
100	3000	$1.6 \cdot 10^3$
200	$300 \cdot 10^3$	$65 \cdot 10^3$
400	$480 \cdot 10^3$	$264 \cdot 10^3$

Tabell 1: *Teoretisk forventa og faktisk antall iterasjoner ved bruk av Jacobis metode. Forventa antall iterasjoner er henta fra [1, p.217]. Vi ser at programmet vårt kjører med færre iterasjoner enn forventa, dette kan ha sammenheng med vår valgte verdi for* $\epsilon = 10^{-8}$*. Hadde vi valgt en mindre* ϵ *så måtte programmet iterert flere ganger for å oppnå ønska presisjon.*

Siden vi vet den analytiske løsninga for en tredimensjonal harmonisk oscillator så kan vi teste programmets presisjon. De tre laveste analytiske egenverdiene er $\lambda_0 = 3$, $\lambda_1 = 7$ og $\lambda_2 = 11$. I tab. 2 er forskjellen mellom analytisk og numerisk verdi for forskjellige n lista opp. Som forventa så blir presisjonen bedre ettersom vi øker n. Valget av $\rho_{\max}$ har også innvirkning på presisjonen, i tab. 3 ser vi på harmonisk oscillator med to elektroner og $\omega_r = 0.01$ og hvordan egenverdiene utvikler seg ettersom $\rho_{\max}$ øker. Potensialet i dette tilfellet er ganske svakt (sammenliknet med de andre tilfellene), så det er naturlig å tenke seg at elektronene vil være lenger fra hverandre. Det betyr at vi må ha en større $\rho_{\max}$ når potensialet er svakt.

n	$\Delta\lambda_0$	$\Delta\lambda_1$	$\Delta\lambda_2$
10	0.14704	0.79008	1.98913
100	0.0092	0.00963	0.0235
200	0.00049	0.00245	0.006
400	0.00012	0.00062	0.0015

Tabell 2: *Differanse mellom analytisk og numerisk verdi for egenverdiene i tilfellet med ett elektron i en harmonisk oscillator. Vi ser at presisjonen blir bedre jo større n er. En annen faktor som har innvirkning på presisjonen er valget av* ρ_{max}*, se tab. 3.*

$\rho_{\max}$	λ_0
10	0.312686
20	0.138030
30	0.108734
50	0.105775
60	0.105774
80	0.105774

Tabell 3: *Programmets stabilitet for egenverdiene i tilfellet med to elektroner i en harmonisk oscillator med* $\omega_r = 0.01$. *Vi ser at presisjonen er avhengig av verdien vi velger for* ρ_{max}. *Velger vi en* ρ_{max} *som er for liten så blir bølgefunksjonen "presset" sammen og resultatet vi får ut blir feil. Vi har her kjørt med* $n = 400$ *for alle verdier av* ρ_{max}.

Resultatene fra Jacobis metode kan vi sammenlikne med Tauts [2] verdier for grunntilstanden med to elektroner og $\omega_r = 0.25$. Taut har i artikkelen sin listet opp egenverdiene ϵ', i starten av artikkelen definerer han $\epsilon' = (1/2)\epsilon$, hvilket betyr at vi må gange tabellverdiene hans med to for å få verdier som stemmer overens med våre resultater. Den analytiske egenverdien blir da $\lambda_{\text{Analytisk}} = 1.25$ I tab. 4 har vi lista opp relativ feil mellom numerisk verdi og analytisk verdi for forskjellige n.

n	Relativ feil
10	0.016
20	0.00468
50	0.00082
100	0.00022
200	0.00006
250	0.00003

Tabell 4: *Differanse mellom numerisk og analytisk verdi av egenverdiene til grunntilstanden i en harmonisk oscillator med to elektroner. Presisjonen er ganske god hele veien, for* $n = 10$ *er feilen på 1.6%, og for større* n *blir den bare mindre og mindre.*

Den numeriske presisjonen avhenger også av vårt valg av ϵ, toleransen for hvor lenge `while`-løkka i funksjonen `JacobiRotation` skal iterere. I tab. 5 har vi lista opp differansen mellom Armadillos

løsning og den fra Jacobis metode for $n = 100$ og $\rho_{\max} = 12$. I den tabellen ser vi at det er en viss forskjell mellom de to numeriske metodene, men at når ϵ blir liten nok så vil forskjellen være så liten at datamaskinen ikke ser forskjell på de.

ϵ	$\lvert\Delta\lambda_0\rvert$
10^{-1}	$8.8 \cdot 10^{-4}$
10^{-2}	$4.47 \cdot 10^{-6}$
10^{-3}	$1.63 \cdot 10^{-7}$
10^{-4}	$4.63 \cdot 10^{-10}$
10^{-5}	$1.30 \cdot 10^{-11}$
10^{-6}	$2.57 \cdot 10^{-13}$
10^{-7}	$2.04 \cdot 10^{-13}$
10^{-8}	$2.04 \cdot 10^{-13}$

Tabell 5: *Differanse mellom Armadillos og Jacobis metodes løsning av laveste egenverdi. For $\epsilon < 10^{-7}$ klarer ikke datamaskinen lenger å gi en nøyaktig differanse mellom resultatene siden de er så like. Til tross for at Jacobis metode er en treg metode, så blir den ganske nøyaktig.*

Konklusjon

Vi har nå løst Schrödingerlikninga for en 3D harmonisk oscillator med både ett og to elektroner. Ved implementering av algoritmen har vi gjort noen observasjoner som er viktige for presisjonen av løsningen vår. Vi har sett at parameterne som har innvirkning på presisjon og stabilitet er n, $\rho_{\max}$ og ϵ. Ved å se på hvilke verdier vi må ha for $\rho_{\max}$ så kan vi justere n deretter. Vi så at algoritmen er ganske presis, men ikke så veldig rask, så vi vil nødig ha en for stor n.

Naturligvis er dimensjonen, n, til matrisen en viktig faktor, her er det viktig at vi velger en dimensjon som er stor nok til at det opprettholder en viss presisjon, men ikke så stor at programmet tar unødig lang tid å kjøre. For at presisjonen til løsningen vår skal være god nok er det også viktig at man velger en maksimumsverdi for distansen mellom elektronene (ρ_{max}) som er stor nok, da det følger at for svakere potensiale må man ha en høyere ρ_{max}, siden elektronene da vil være lengre fra hverandre. Også valget av ϵ har en betydning for presisjonen, da dette er toleransen i `while`- løkken vår. For at vi skal ha et program som fungerer slik vi ønsker, er

det følgelig viktig å teste programmet og endre på variabler som har betydning på presisjonen. Å utføre enhetstester er da en viktig faktor for å sikre seg et program som gir en god tilnærming. Vi konkluderer med at stryken på potensiale i en harmonisk oscillator har mye å si for hvor elektronene er, og at Jacobis metode er en nøyaktig, men treg metode.

Referanser

[1] Morten Hjorth-Jensen, *Computational Physics, Lecture Notes Fall 2015*

[2] M. Taut, Physical Review A, 3561 - 3566 (1993)