FYS3150 Prosjekt 2

Øyvind Are Rysstad Johnsen og Thomas Haaland

04.10.2015

I dette prosjektet skal vi løse matriselikningen

$$\mathbf{Au} = \lambda \mathbf{u} \tag{1}$$

som også kan skrives som

$$\begin{pmatrix} d_1 & e_1 & 0 & 0 & \dots & 0 & 0 \\ e_1 & d_2 & e_2 & 0 & \dots & 0 & 0 \\ 0 & e_2 & d_3 & e_3 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & \dots & \dots & \dots & d_{n_{step}-2} & e_{n_{step}-1} \\ 0 & \dots & \dots & \dots & \dots & e_{n_{step}-1} & d_{n_{step}-1} \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ \vdots \\ \vdots \\ u_{n_{step}-1} \end{pmatrix} = \lambda \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ \vdots \\ \vdots \\ u_{n_{step}-1} \end{pmatrix} \tag{2}$$

Dette gjør vi ved å benytte oss av en rotasjonsmatrise slik at

$$\mathbf{B} = \mathbf{S^{-1}AS} \tag{3}$$

der $\mathbf{S}$ blir brukt på $\mathbf{A}$ slik at $\mathbf{B}$ er en diagonalmatrise hvor diagonalmatrise elementene er egenverdiene λ_i. Matrisen $\mathbf{S}$ inneholder $\cos\theta$ og $\sin\theta$ slik at det elementet som som det roteres rundt blir 0, som beskrevet i oppgaveteksten. Dette gir oss likningen

$$t = -\tau \pm \sqrt{1+\tau^2} \tag{4}$$

der $t = \tan\theta$ og $\tau = \cot 2\theta = \frac{a_{ll}-a_{kk}}{2a_{kl}}$. Vi velger den miste roten, siden det innebærer at $|\theta|$ blir minst mulig, siden det innebærer at $|\tan\theta|$ blir mindre enn 1 og $-\frac{\pi}{4}<\theta<\frac{\pi}{4}$. Siden vi finner $\cos\theta$ ved hjelp av

$$c = \frac{1}{\sqrt{1+t^2}} \tag{5}$$

vil vi at t skal være nær 0 for for at c skal være nær 1 slik at vi minimerer forskjellen mellom matrisene $\mathbf{A}$ og $\mathbf{B}$ som vi ser av

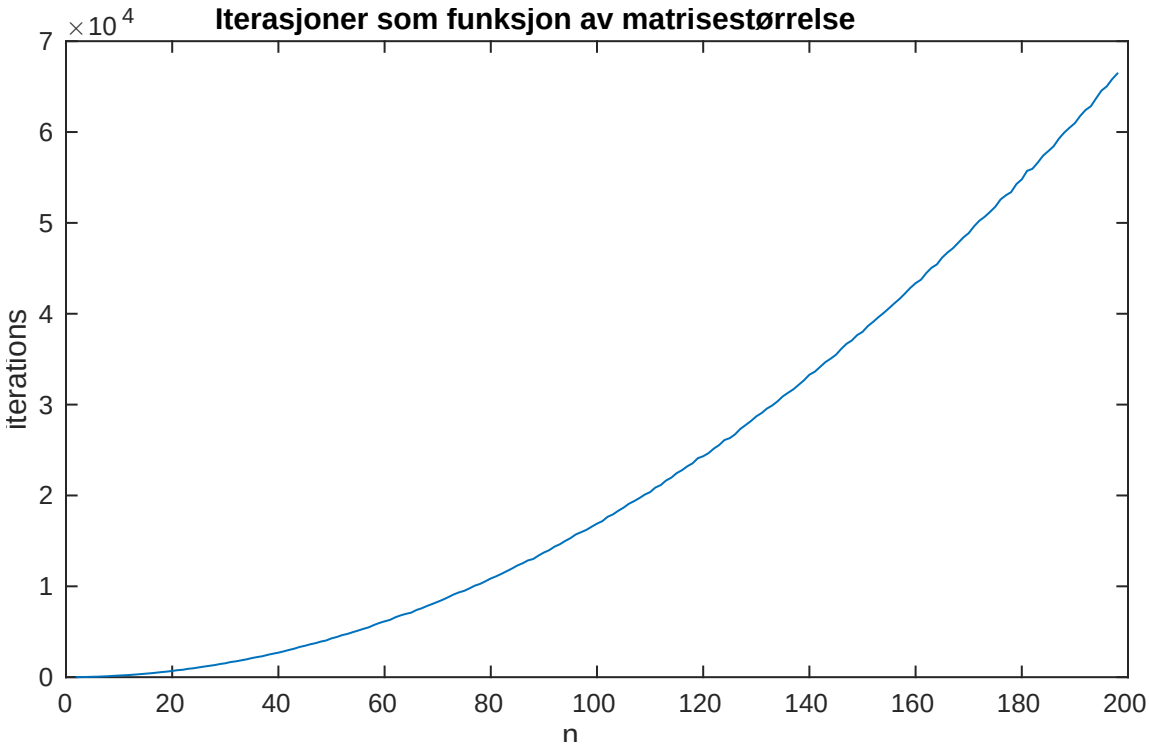
$$||\mathbf{B}-\mathbf{A}||_F^2 = 4(1-c)\sum_{i=1, i\neq k,l}^{n}\left(a_{ik}^2 + a_{il}^2 + \frac{2a_{kl}^2}{c^2}\right) \tag{6}$$

Vi implementerer Jacobi algoritmen i et C++ program og finner at $n_{step} = 150$ er laveste oppløsning der får vi en nøyaktighet på 4 ledende siffer. Vi må da benytte en $\rho_{max} = 4.37$. Algoritmen bruker 37652 iterasjoner med dette valget, som er det som trengs for å sette alle ikkediagonale elementer til mindre enn toleransen $\epsilon = 10^{-8}$. Når vi plotter antall iterasjoner som en funksjon av matrisens dimensjon ser det ut som at stegantallet øker kvadratisk. Vi ser fra figurene at økningen iterasjoner er kvadratisk med n. Når vi sammenligner med Jacobi metoden med Armadillos innebygde matriseløser ser vi at sistnevnte bruker vesentlig mindre tid. For $n = 150$ brukte Armadillo $7.2 \times 10^{-3}s$ mens Jacobi metoden brukte $6.6 \times 10^{-1}s$.

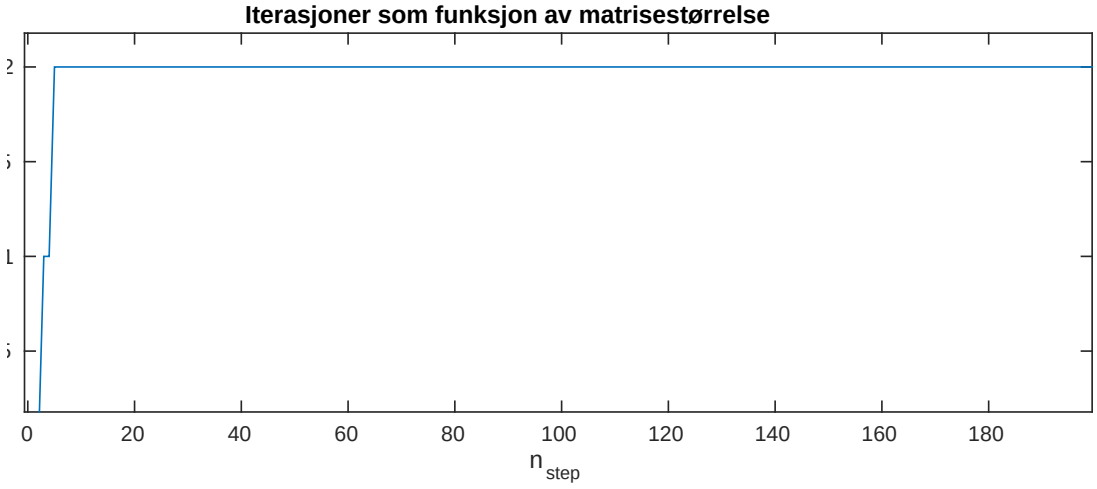
Vi implementerer og løser Schrødingerlikningen for to elektroner i samme potensial, uten frastøtende Coloumb-vekselvirkninger for forskjellige ω_r. Vi ser at for $\omega_r = 1$ er grunntilstanden for ett elektron og to elektroner nesten identisk. Det er en liten forskjell som kommer av $\frac{1}{\rho}$.

$$-\frac{d^2}{d\rho^2}\psi(\rho) + \omega_r^2\rho^2\psi(\rho) + \frac{1}{\rho} = \lambda\psi(\rho) \tag{7}$$

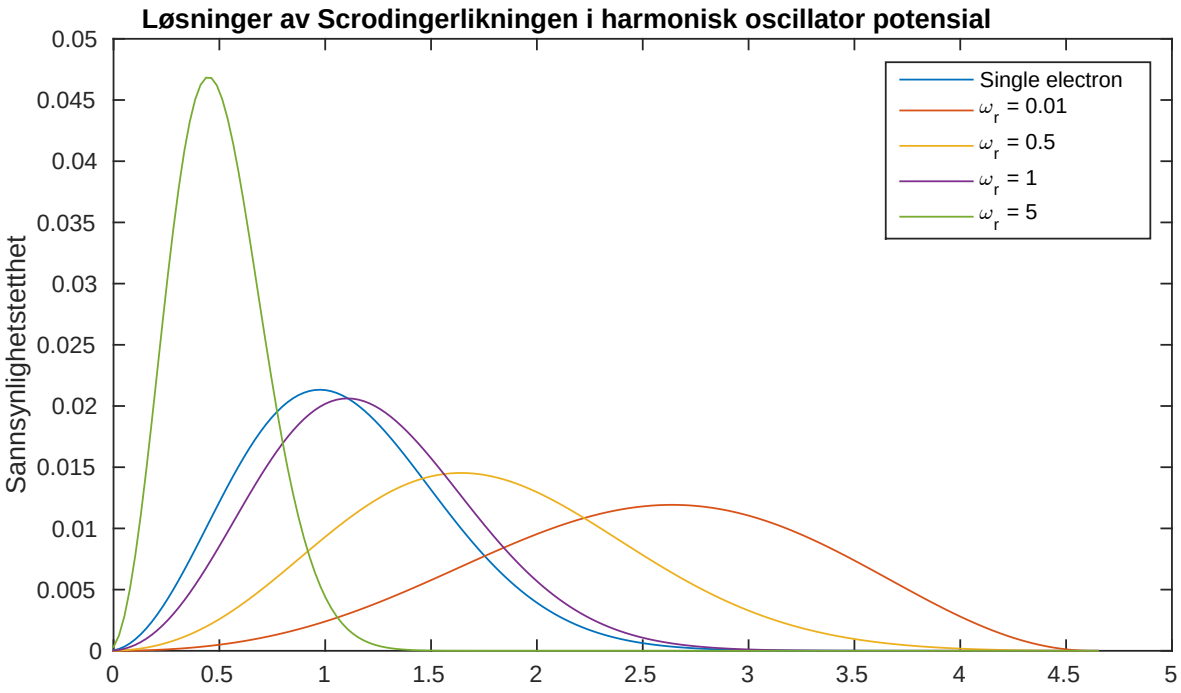
ω_r kan vi se på som et mål på styrken av potensialet. For større ω_r ser vi at grunntilstanden blir skjøvet inn mot sentrum og blir skarpere definert enn for mindre verdier.



Figur 1: Når vi teller antall iterasjoner som funksjon av matrisestørrelse virker det som at antall iterasjoner går som n^2.



Figur 2: For hver matrisestørrelse n bruker vi $\log_n$ på antall iterasjoner, og får at antall iterasjoner går som n^2 nøyaktig.



Figur 3: Når vi plotter bølgefunksjonen for ett enkelt elektron sammen med bølgefunksjen for to elektroner med forskjellig styrke i potensialet ser vi at bølgefunksjonene likner i grunntilstanden.

Når vi kjører programmet får vi utskriften nedenfor i terminalvinduet.

⋮

48

49

50

C: 2.99974, 6.999, 11.009

Victory!

Test results:

Mmax is working as advertised!

Rotate is working as advertised!

Jacobi_eigenvalues is working as advertised!

Successful tests: 3 out of 3 tests performed!

Resultatene våre viser at Jacobi-rotasjonsalgoritmen er mindre effektiv med tidsbruken enn tridiagonalløsningsmetoden fra forrige prosjekt. Jacobi-rotasjonsalgoritmen bruker n^2 iterasjoner og hver iterasjon brukes $\simeq 6n$ FLOPS som gir $6n^3$ FLOPS. Men, vi ser også at Jacobi-rotasjonsalgoritmen bruker mange if/else tester. Med QtCreator sin innebygde funksjonanalyse finner vi at funksjonen Mmax står for 94.3% av belastningen i programmet. Vi ser dermed at det er denne delen av programmet vi burde effektivisere. I stedet kunne vi gjennomført rotering på alle elementer i matrisen vi skal løse for, og etterpå sjekket om alle offdiagonale elementer er null. Vi forventer at tidsbruken ville gått betydelig ned.

Githubadresse for kode:
https://github.com/thomashaaland/Fys4150/blob/master/prosjekt2/main.cpp