CHALMERS

Emulering av deuteronens bindningsenergi med avseende på pionmassan

Kandidatarbete inom Fysik

Arvid Kolstad
Edward Fritzell
Deniz Atay
Axel Wijk

INSTITUTIONEN FÖR FYSIK

CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2025
www.chalmers.se

Kandidatarbete 2025

Emulering av deuteronens bindningsenergi med avseende på pionmassan

Emulation of the deuteron binding energy with respect to the pion mass

ARVID KOLSTAD, EDWARD FRITZELL, DENIZ ATAY,
AXEL WIJK.

Instutitionen för fysik
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sweden 2025

Emulering av deuteronens bindningsenergi med avseende på pionmassan.
ARVID KOLSTAD, EDWARD FRITZELL, DENIZ ATAY, AXEL WIJK.

© ARVID KOLSTAD, EDWARD FRITZELL, DENIZ ATAY, AXEL WIJK, 2025.

Handledare: Andreas Ekström , Institutionen för fysik vid Chalmers tekniska högskola

Examinator: Magnus Hörnqvist Colliander, Institutionen för fysik vid Chalmers tekniska högskola

Kandidatarbete 2025
Chalmers tekniska högskola
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Skriven i LATEX
Göteborg, Sverige 2025

Emulering av deuteronens bindningsenergi med avseende på pionmassan

ARVID KOLSTAD, EDWARD FRITZELL, DENIZ ATAY, AXEL WIJK
Institutionen för fysik
Chalmers tekniska högskola

SAMMANDRAG

Detta arbete undersöker hur deuteronens bindningsenergi påverkas av variationer i pionmassan, samt hur detta beroende effektivt kan emuleras med numeriska metoder. Deuteronen, som består av en proton och en neutron, hålls samman av den starka kärnkraften, vilken modelleras genom Yukawa-potentialen där pionmassan har en central roll.

För att minska beräkningskostnaden och simuleringstiden för detta icke-linjära problem används en kombination av två metoder: den Reducerade Basmetoden (RBM) och den Empiriska Interpolationsmetoden (EIM). RBM reducerar dimensionskostnaden genom att projicera beräkningarna till ett underrum, men kräver linjärt parameterberoende. Därför används EIM för att approximera det icke-linjära beroendet av pionmassan som en linjär summa.

Metoderna har implementerats i Python och utvärderats med avseende på noggrannhet och prestanda. Resultaten visar att emuleringen med RBM och EIM uppnår god överensstämmelse med fullständiga simuleringar, men med avsevärt lägre beräkningskostnad. Detta gör metoden särskilt relevant för tyngre kärnsystem där direkt simulering är beräkningsmässigt opraktisk.

ABSTRACT

This thesis investigates how the binding energy of the deuteron is affected by variations in the pion mass, and how this dependence can be efficiently emulated using numerical methods. The deuteron, consisting of one proton and one neutron, is bound by the strong nuclear force, which is modeled using the Yukawa potential, where the pion mass plays a central role.

To reduce computational cost and simulation time for this nonlinear problem, a combination of two methods is employed: the Reduced Basis Method (RBM) and the Empirical Interpolation Method (EIM). RBM reduces the dimensional complexity by projecting calculations onto a subspace, but assumes linear parameter dependence. Therefore, EIM is used to approximate the nonlinear pion mass dependence as a linear sum.

These methods were implemented in Python and evaluated in terms of accuracy and performance. The results show that the combined RBM and EIM approach achieves high agreement with full simulations while significantly reducing computational cost. This makes the approach particularly useful for heavier nuclear systems, where direct simulation is computationally impractical.

The report is written in Swedish.

Nyckelord: Deuteronen, bindningsenergi, pionmassa, EIM, RBM.

Innehåll[1]

[1]I detta arbete har en AI används för att förädla språkbruket. Detta har involverat prompts om språkbruk och ordval. Ingen fakta har hämtats från denna AI, utan denna kommer enbart från källorna i referensavsnittet.

1 Introduktion

Under lång tid betraktades atomen som materiens minsta odelbara beståndsdel. Denna föreställning började ifrågasättas i början av 1900-talet, delvis genom Ernest Rutherfords experiment med alfapartiklar, som visade att atomen i själva verket har en inre struktur med en kärna [1]. Denna upptäckt var del av ett paradigmskifte inom fysiken och bidrog till utvecklingen av kvantmekaniken, som ersatte klassiska beskrivningar med sannolikhetsbaserade modeller. Kärnan består av så kallade nukleoner, ett samlingsord för protoner och neutroner, som hålls samman av den starka kraften – en av naturens fyra fundamentala krafter. Till skillnad från den elektromagnetiska kraften verkar den starka växelverkan endast över mycket korta avstånd, och dess mikroskopiska natur kan inte förklaras av klassisk fysik. En tidig teoretisk beskrivning gavs av Hideki Yukawa, som 1935 föreslog att kraften mellan nukleoner förmedlas av en ny partikel – pionen (π)[2]. Detta var den första framgångsrika modellen för kärnkraftens räckvidd och styrka.

Pionen är en typ av partikel som spelar en viktig roll i atomkärnans sammanhållning. Pioner är instabila partiklar som spelar en central roll i teorier om den starka kärnkraften. Där förekommer de som kortlivade partiklar som snabbt uppstår och försvinner igen när krafter utväxlas mellan protoner och neutroner. Den kraft som förmedlas genom detta utbyte kan beskrivas med så kallade Yukawapotentialer, där räckvidden är omvänt proportionell mot pionens massa. I en kvantmekanisk modell av deuteronen – den enklaste bundna atomkärnan bestående av en proton och en neutron – ingår Yukawa-potentialen som beror icke-linjärt på pionens massa enligt:

$$V_{1\pi} \propto \frac{1}{q^2 + m_\pi^2},$$

där q är överförd rörelsemängd [3]. Variationer i pionens massa påverkar därmed den attraktiva kraften mellan nukleoner och i förlängningen bindningsenergin. Att studera detta beroende är intressant ur både kärnfysikalisk och metodologisk synvinkel, eftersom det bidrar till vår förståelse av kärnstruktur och hur den påverkas av fundamentala parametrar.

För tyngre atomkärnor ökar dock den numeriska komplexiteten för att lösa Schrödingerekvationen avsevärt. Beräkningskostnaden växer exponentiellt med antalet nukleoner i systemet, vilket gör direkta simuleringar - numeriska lösningar av de fysiska modellerna - där olika potentialparametrar varieras opraktiska. För att kunna genomföra sådana studier används därför ofta olika typer av emuleringsmetoder, där en approximativ modell tränas utifrån ett antal träningspunkter och sedan används för att förutsäga resultat för nya parametrar. Detta förenklar beräkningarna utan att offra för mycket precision och noggrannhet. En effektiv sådan metod är den reducerade basmetoden (RBM), som dock förutsätter ett linjärt parameterberoende. Pionmassans inverkan på potentialen är icke-linjär och RBM kan ej appliceras direkt. Detta problem kan lösas genom att kombinera RBM med den empiriska interpolationsmetoden (EIM), som möjliggör en approximation av den icke-linjära potentialen som en summa av linjära termer. På så vis kan RBM tillämpas för att emulera deuteronens bindningsenergi med avseende på pionmassan.

Syftet med detta projekt är att implementera och analysera huruvida RBM och EIM kan minska beräkningskostnaden utan att förlora nämnvärd precision och för att kartlägga möjligheten för emulering för större och mer komplex kärnsystem med avseende på variationer i pionmassan.

2 Teori

För att förstå de problemställningar och metoder som behandlas i detta arbete krävs insikter inom både kvantmekanik och numeriska metoder. Detta kapitel ger en översikt över den teori och de metoder som utgör grunden för projektet.

2.1 Deuteronen och kvantmekanik

Inom klassisk fysik är det möjligt att veta en partikels position, x, och därigenom partikelns hastighet, v, genom tidsderivatan av positionen, $v = \frac{dx}{dt}$. Detta gör det möjligt att veta både partikelns position och hastighet samtidigt. I kvantmekaniska system är det däremot omöjligt att veta partikelns position och hastighet samtidigt utan att det finns en osäkerhet. Denna osäkerhet är beskriven av Heisenbergs osäkerhetsprincip:

$$\frac{\hbar}{2} \leq \Delta x \Delta p, \tag{1}$$

där $\hbar$ är Plancks konstant delat med 2π, Δx och Δp är osäkerheten i position respektive rörelsemängd [4]. Ju noggrannare man bestämmer partikelns position, desto större blir osäkerheten i dess hastighet, och vice versa. För att beskriva kvantmekaniska system har man introducerat en formalism för att ta hänsyn till osäkerheten.

Inom kvantmekaniken är vågfunktionen en lösning till Schrödingerekvationen, vars roll förtydligas i avsnitt 2.1.1, och beskriver det kvantmekaniska tillståndet hos ett system, exempelvis partikelns energi eller banrörelsemoment. I detta arbete betecknas vågfunktionen med $\psi(p,t)$ och är en komplexvärd funktion av rörelsemängd och tid. Enligt Max Borns tolkning är beräkning av vågfunktionen sådan att kvadraten av absolutbeloppet av vågfunktionen, $|\psi(p,t)|^2$, ger sannolikhetstätheten att finna partikeln med en viss rörelsemängd, p, vid tid, t. För att sannolikhetstolkningen ska vara konsekvent måste vågfunktionen vara normerad, enligt följande:

$$\int_{-\infty}^{\infty} |\psi(p,t)|^2 \, dp = 1. \tag{2}$$

För att beskriva mätbara storheter inom kvantmekaniken används hermitska operatorer: matematiska verktyg som liknar funktioner och talar om hur ett kvantmekaniskt system påverkas av en viss mätning. En operator, $\hat{Q}$, associeras med ett egenvärde, q, genom en egenvärdesekvation där ψ motsvarar egenfunktionen till operatorn $\hat{Q}$ enligt:

$$\hat{Q}\psi = q\psi. \tag{3}$$

Om $\hat{Q}$ är en hermitsk operator uppfyller den att $\hat{Q} = \hat{Q}^\dagger$, där $\hat{Q}^\dagger$ är adjunkten till $\hat{Q}$, och likheten medför att operatorns egenvärden är reella. Detta innebär att egenvärdet, q, kan representera en fysikalisk storhet: till exempel energi eller rörelsemängdsmoment, och vågfunktionen, ψ, kan förknippas med en partikels sannolikhet att besitta egenvärdet q. Den hermitska operatorn som associeras med systemets totala energi kallas för hamiltonianen och betecknas $\hat{H} \equiv \hat{T} + \hat{V}$, där $\hat{T}$ är systemets kinetiska energi och $\hat{V}$ är systemets potentiella energi. Hamiltonianen ligger som grund för arbetet och behandlas först i teoriavsnittet 2.1.1.

Ännu en faktor att beakta, som introducerar ny formalism, är att vågfunktioner till bundna tillstånd lever i Hilbertrummet $\mathcal{L}^2(a,b)$, som medför att de är kvadrat-integrerbara:

$$\int_a^b |\psi|^2 dp < \infty, \ \int_a^b |\phi|^2 dp < \infty, \tag{4}$$

och det existerar en inre produkt som i Diracnotation skrivs som:

$$\langle \phi \, | \psi \rangle = \int_a^b (\phi^* \psi) dx, \tag{5}$$

där ϕ^* är det komplexa konjugatet till ϕ. En vektor i Hilbertrummet beskrivs, i Diracnotation, av en ket: $|\psi\rangle$, och det adjungerade tillståndet skrivs som en bra: $\langle\psi|$. Den inre produkten $\langle\phi\,|\psi\rangle = \alpha$ tolkas som en sannolikhetsamplitud, α, vars absolutbelopp i kvadrat, $|\alpha|^2$, beskriver sannolikheten att systemet vid mätning mäts vara i tillståndet $|\phi\rangle$ om systemet tidigare befann sig i tillståndet $|\psi\rangle$.

När partikeln endast kan befinna sig i tillstånd som, till antalet, kan approximeras som ändliga och diskreta kan operatorerna och vågfunktionerna representeras som matriser respektive vektorer; vilket är fallet för detta arbete. Denna matris-vektornotation tillåter beräkningar att utföras numeriskt som är till fördel vid datorsimuleringar och -emuleringar. Matris-vektornotationen innebär att den inre produkten mellan två vågfunktioner, $|\phi\rangle$ och $|\psi\rangle$, blir som följande:

$$\langle\phi\,|\psi\rangle = \sum_{i=1}^{n}\langle\phi\,|i\rangle\,\langle i\,|\psi\rangle\,, \tag{6}$$

där n är antalet tillstånd och $|i\rangle$ är basen som valdes för att representera $\langle\phi|$ och $|\psi\rangle$.

I kvantmekaniska system är det vanligt att räkna i rörelsemängdsbasen som medför att vågfunktionen ges av $\psi(p,t) = \langle p\,|\psi(t)\rangle$, där $|p\rangle$ är egenfunktionen till rörelsemängdsoperatorn, $\hat{P}$. I rörelsemängdsrummet kan en potential, $\hat{V}$, uttryckas på matrisform som $\hat{V}_{mn}(p',p) = \langle p'_m|\,\hat{V}\,|p_n\rangle$, där p'_m och p_n beskriver de rörelsemängdstillstånd som partikeln kan besitta och $\hat{V}_{mn}(p',p)$ betecknar potentialens element i rad m och kolonn n. I det här arbetet kommer det huvudsakligen att arbetas i rörelsemängdsrummet.

I kvantfysiken har partiklar, förutom ett banrörelsemängdsmoment, även ett kvantmekaniskt spinn; som är en storhet som beskriver partikelns intrinsiska rörelsemängdsmoment. En partikels kvantiserade nivåer för banrörelsemängdsmoment betecknas med l medan spinnet betecknas med s. Det totala rörelsemängdsmomentet betecknas med j och ges av en kopplad, total rörelsemängdsmomentsoperator, $\hat{J}$ enligt

$$\hat{J} = \hat{L} + \hat{S}, \tag{7}$$

där $\hat{L}$ och $\hat{S}$ är banrörelsemängdsmomentsoperatorn respektive spinnoperatorn och ekvationen 7 medför att $|l-s| \leq j \leq |l+s|$. I det här arbetet kommer kvanttal skrivna med gemener: j, l, s, att referera till en nukleons enskilda tillstånd medan kvanttal skriven med versaler: J, L, S, beskriver det relativa proton-neutron tillståndet, det vill säga, deuteronens kvanttal relativt dess masscentrum.

Deuteronen är ett bundet tillstånd av en proton och en neutron vilket medför att de ingående nukleonerna har diskreta energinivåer. Deuteronen har endast ett bundet tillstånd med $J = 1$ med positiv paritet. Paritet är en storhet som är bevarad i den starka kraften och ges av $(-1)^L$, där L är kvanttalet för systemets banrörelsemängdsmomentet. Ännu en faktor att beakta är att nukleonerna är spinn-halv partiklar med kvanttal $s = \frac{1}{2}$, som innebär att deuteronens spinnkvanttal kan vara $S = 0, 1$, beroende på om neutronens och protonens spinn är anti-parallella respektive parallella. Att deuteronens enda bundna tillstånd kräver att $J = 1$, att S kan vara högst 1 och en positiv paritet begränsar deuteronens kvanttal för banrörelsemängdsmomentet till $L = 0$ eller $L = 2$; vilket bestämdes med hjälp av ekvation 7. Spinntillståndet $S = 0$ är inte möjlig då $J = 1$ eftersom detta skulle medföra att $L = 1$ som resulterar i en negativ paritet. Deuteronens möjliga tillstånd kan beskrivas som ${}^3S_1$ och ${}^3D_1$ enligt spektroskopisk notation: ${}^{2S+1}L_J$, där S och D betecknar $L = 0$ respektive $L = 2$. Med deuteronens möjliga tillstånd fastställda kan den fullständiga vågfunktionen för deuteronen skrivas som en superposition av ${}^3S_1$ och ${}^3D_1$ tillstånden[5]:

$$|\psi\rangle = a\,|{}^3S_1\rangle + b\,|{}^3D_1\rangle\,, \tag{8}$$

där a och b är konstanter som bestämmer sannolikhetsförhållandet mellan de två möjliga tillstånden och är sådana att vågfunktionen är normerad: $\langle\psi\,|\psi\rangle = 1$.

2.1.1 Hamiltonianen för en deuteron

För att ta fram deuteronens bindningsenergi kan man använda Schrödingerekvationen i dess tidsoberoende form, vilket är fundamental inom kvantmekanik och beskriver tillstånd och energinivåer för ett stationärt system: ett system vars egenfunktioner och egenvärden inte ändras över tid, vilket gäller för energiegentillstånd [4]. Den ges av:

$$\hat{H}\,|\psi\rangle = E\,|\psi\rangle\,, \tag{9}$$

där $\hat{H}$ är hamiltonoperatorn och innehåller information om deuteronens totala energi, $|\psi\rangle$ representerar systemets kvanttillstånd och E är systemets energi. Hamiltonianen, $\hat{H}$, består av två olika bidrag: kinetiska energin $\hat{T}$ och potentiella energin $\hat{V}$, vilket ger följande uttryck:

$$\hat{H} = \hat{T} + \hat{V}. \tag{10}$$

Potentialen i uttrycket för hamiltonianen, ekvation 10, ges av den starka kraften mellan protonen och neutronen och kan delas upp i olika bidrag. En viktig komponent i potentialen för den starka nukleonkraften är EPU (En-Pion-Utbyte) som uppstår genom utbyte av en pion mellan nukleonerna[3]. I potentialen ingår även så kallade kontakttermen, V_{ct}, som beskriver växelverkan mellan nukleoner på mycket kort avstånd. Potentialen i rörelsemängdsrummet ges därför av:

$$\hat{V} = V_{ct}(\vec{p}', \vec{p}) + V_{1\pi}(\vec{p}', \vec{p}), \tag{11}$$

där $\vec{p}'$ och $\vec{p}$ betecknar den slutliga respektive initiala rörelsemängden mellan två nukleoner. Kontaktpotentialens termer går att bryta ner ytterligare. V_{ct}-termen har ett linjärt beroende av två parametrar enligt:

$$V_{ct}(\vec{p}', \vec{p}) = C_S + C_T \vec{\sigma}_1 \cdot \vec{\sigma}_2, \tag{12}$$

där parametern C_S beskriver kontakt för $S = 0$ och C_T för $S = 1$, $\vec{\sigma}_1$ och $\vec{\sigma}_2$ är spinnoperatorerna för nukleon 1 respektive 2. På grund av deuteronen har kvanttal $S = 1$ är det endast parametern C_T som spelar en roll. Potentialen för EPU har däremot ett icke-linjärt parameterberoende av pionmassan och ges av:

$$V_{1\pi}(\vec{p}', \vec{p}) = -\frac{g_A^2}{4f_\pi^2} \vec{\tau}_1 \cdot \vec{\tau}_2 \frac{\vec{\sigma}_1 \cdot \vec{q} \vec{\sigma}_2 \cdot \vec{q}}{q^2 + m_\pi^2}, \tag{13}$$

där $\vec{q} = \vec{p}' - \vec{p}$, $\vec{\tau}_1$ och $\vec{\tau}_2$ är isospinnoperatorerna och f_π och g_a är kopplingskonstanter som karaktäriserar hur starkt pioner och nukleoner växelverkar. Eftersom bindingsenergin undersöks som funktion av pionmassan är endast följande, icke-linjära proportionalitet av intresse:

$$V_{1\pi}(\vec{p}', \vec{p}) \propto \frac{1}{q^2 + m_\pi^2}. \tag{14}$$

Den kinetiska energin för deuteronen kan uttryckas som: $\hat{T} = \frac{\hat{P}^2}{2\mu}$, där $\hat{P}$ är den relativa rörelsemängdsoperatorn och μ är den reducerade massan. Med känd kinetisk energi kan hamiltonianen uttryckas som:

$$\hat{H} = \frac{\hat{P}^2}{2\mu} + \hat{V}. \tag{15}$$

Om även den potentiala energin är känd kan Schrödingerekvationen lösas genom att representera hamiltonianen på matrisform i rörelsemängdsrummet och därefter diagonalisera den för att hitta egenvärdena. Matrisrepresentationen fås enligt följande[6]:

$$\langle p', L' | \hat{H} | p, L \rangle = \frac{p^2}{2\mu} \delta_{L'L} + V_{L'L}(\vec{p}', \vec{p}), \tag{16}$$

där $\delta_{L'L}$ är Kroneckers delta som uppstår då den kinetiska energin är diagonal i rörelsemängdsrummet och $L', L \in \{S, D\}$ betecknar det slutliga respektive initiala banrörelsemängdsmomentet. Som konsekvens av vågfunktionens utseende, som syns i ekvation 8, får hamiltonmatrisen en blockstruktur med fyra typer av element som motsvarar alla möjliga tillstånd deuteronen kan ha innan och efter nukleonernas växelverkan, enligt:

$$\hat{H}(\vec{p}', \vec{p}) = \begin{pmatrix} \langle p', S | \hat{H} | p, S \rangle & \langle p', S | \hat{H} | p, D \rangle \\ \langle p', D | \hat{H} | p, S \rangle & \langle p', D | \hat{H} | p, D \rangle \end{pmatrix} \tag{17}$$

2.1.2 Regulering av potentialen

För att säkerhetsställa att deuteronens hamiltonian kan implementeras numeriskt måste potentialen, $\hat{V}$, reguleras eftersom beräkning av potentialen i rörelsemängdsrummet divergerar för större rörelsemängder. Detta uppnås genom att multiplicera potentialen med en regulatorfunktion $f(p', p)$, som dämpar bidrag från momenta som ligger bortom teorins giltighetsområde[3]. Detta ger

$$\hat{V}(\vec{p}', \vec{p}) \to \hat{V}(\vec{p}', \vec{p}) f(p', p), \tag{18}$$

där regulatorfunktionen ges av:

$$f(p', p) = \exp\left[-\left(\frac{p'}{\Lambda}\right)^{2n} - \left(\frac{p}{\Lambda}\right)^{2n} \right]. \tag{19}$$

Här är Λ cutoff-parametern, som typiskt ligger omkring $500\,\mathrm{MeV}/c_0$, och n styr hur snabbt regulatorn dämpar höga momenta. Regulatorn påverkar inte den lågenergifysik som teorin är avsedd att beskriva, eftersom $f(p', p) \approx 1$ när $p, p' \ll \Lambda$. Introducering av cutoff-parametern tar inte heller bort någon precision i resultatet eftersom reguleringen som sker för EPU tas hänsyn till genom kontakttermen. För vidare läsning om den fysikaliska motivationen bakom regulatorn, se [3].

Cut-off parametern Λ kan varieras vilket påverkar intervallet av intressant momenta. Däremot i detta arbete är inget särskilt intervall av större vikt och därför ansätts det typiska värdet $\Lambda = 500\,\mathrm{MeV}/c_0$.

2.2 Gauss-Legendre kvadratur

För att beräkna deuteronens potential i rörelsemängdsrummet kommer en integral behöva utfärdas över de möjliga värdena på de relativa rörelsemängderna mellan de ingående nukleonerna i atomkärnan. Om man från Schrödingerekvationen endast beaktar kontakttermerna och potentialtermen för EPU fås följande integral [7]:

$$\int_0^\infty \langle p', L' | \hat{V} | p, L \rangle \langle p, L | \psi \rangle p^2 dp = \int_0^\infty \langle p', L' | \left(V_{ct}(\vec{p}', \vec{p}) + V_{1\pi}(\vec{p}', \vec{p})\right) | p, L \rangle \langle p, L | \psi \rangle p^2 dp, \tag{20}$$

där $|p', L'\rangle$ är deuteronens slutliga tillstånd och $|p, L\rangle$ är deuteronens initiala tillstånd. På grund av cutoff-parametern, som introduceras i avsnitt 2.1.2, avtar potentialen exponentiellt och integralen kan därför approximativt uttryckas över ett begränsat integrationsintervall:

$$\int_0^\infty \langle p', L' | \hat{V} | p, L \rangle \langle p, L | \psi \rangle p^2 dp \approx \int_0^d \langle p', L' | \hat{V} | p, L \rangle \langle p, L | \psi \rangle p^2 dp, \tag{21}$$

där $d = 1000$ MeV/c beskriver en finit rörelsemängd. Eftersom emuleringen av deuteronens bindningsenergier sker med hjälp av digitala verktyg måste integralerna approximeras med summor över ett begränsat summeringsintervall. Gauss-Legendre kvadratur är en metod för att approximera ändliga integraler över integrationsintervallet $[-1, 1]$ och använder sig av Legendre polynom [6] som, av Rodrigues formel, ges på formen:

$$P_n(x) = \frac{1}{2^n n!} \frac{d^n}{dx^n}(x^2 - 1)^n, \tag{22}$$

där P_n är den n:te Legendre polynomet. För en integrerbar och ändlig funktion $f(x)$ beroende på variabeln x kan Gauss-Legendre kvadraturen beskrivas på följande form:

$$\int_{-1}^{1} f(x) dx \approx \sum_{i=1}^{n} w_i f(x_i), \tag{23}$$

där x_i motsvarar nollställena för n:te Legendre polynomet, w_i är kvadraturvikter och n är antalet samplingspunkter med lika många punkter som graden till Legendre polynomet. Den i:te kvadraturvikten kan uttryckas som:

$$w_i = \frac{2}{(1 - x_i^2)(P_n'(x_i))^2}, \tag{24}$$

där P_n' är den n:te Legendre polynomets derivata. För att integrera över ett godtyckligt, ändligt intervall kan ett variabelbyte göras och integralen kan uttryckas med Gauss-Legendre kvadratur:

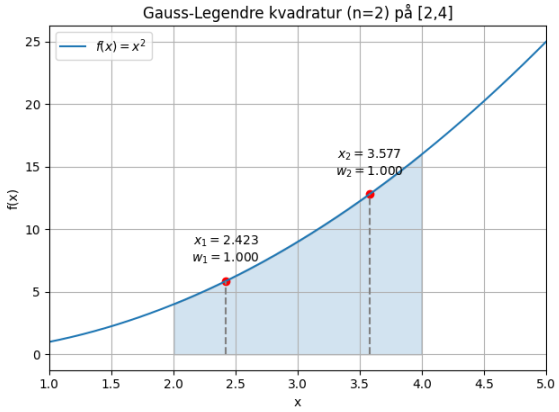
$$\int_a^b f(t)dt = \frac{b-a}{2}\int_{-1}^{1} f(\frac{b-a}{2}x + \frac{b+a}{2})dx \approx \frac{b-a}{2}\sum_{i=1}^{n} w_i f(\frac{b-a}{2}x_i + \frac{b+a}{2}), \tag{25}$$

där det utförde variabelbytet är $t = \frac{b-a}{2}x + \frac{b+a}{2}$ där a och b är godtyckliga, finita tal. Gauss-Legendre kvadratur med n samplingspunkter ger exakta lösningar för integraler av polynom av grad $2n-1$.

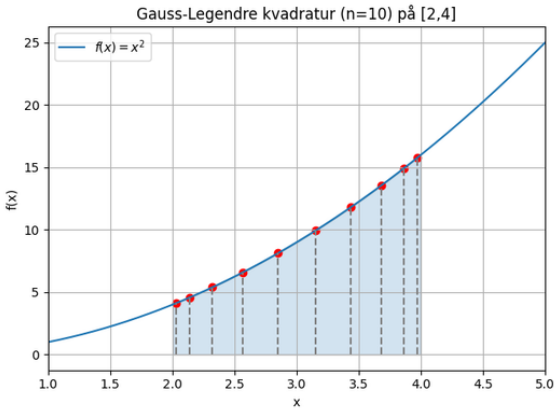
För att illustrera hur Gauss–Legendre-kvadratur fungerar i praktiken visas nedan de vikter och punkter som används för att approximera följande integral:

$$\int_2^4 x^2 dx \approx 18,667. \tag{26}$$

Från figurerna ser vi att ett polynom av grad två kan lösas med Gauss-Legendre kvadratur med $n = 2$ samplingspunkter samt hur fördelningen av kvadraturpunkter ser ut för större n.



Figur 1: Lösning av $\int_2^4 x^2\,dx$ med Gauss-Legendre kvadratur för $n = 2$ punkter. De två punkterna och deras vikter visualiserar hur metoden fungerar även med få punkter. Approximationen ger att integralens värde är $2,423^2 + 3,577^2 \approx 18,666$ där felet kan förklaras med numeriska avrundningar.



Figur 2: Lösning av $\int_2^4 x^2\,dx$ med Gauss-Legendre kvadratur för $n = 10$ punkter. Här syns hur punkterna fördelas tätare mot ändarna av intervallet, vilket återspeglar metodens anpassning till funktionens form.

2.3 Algoritmer och numeriska metoder för emulatorn

I detta avsnitt kommer teorin för utvecklingen av emulatorn att presenteras och förklaras. Först kommer RBM-metoden att förklaras då detta är kärnan i emuleringsprocessen. Därefter beskrivs EIM-approximeringen, för att hantera det olinjära sambandet mellan deuteronens bindningsenergi och pionmassan. Till sist kommer den valda interpoleringsmetoden att förklaras samt den beräkningskostnad som krävs för simulering kontra emulering.

2.3.1 Reducerad Basmetod

För att reducera beräkningskostnaden av att diagonalisera deuteronens hamiltonian, $\hat{H}$, och att ta fram bindningsenergin som funktion av pionmassan, m_π, ska RBM (Reducerad Basmetod) användas som möjliggör att projicera större matriser på underrum [8]. För att utföra RBM på en $m \times m$ matris, $M(\alpha)$, förutsätts ett affint parameterberoende så att $M(\alpha)$ kan skrivas som:

$$M(\alpha) = M_0 + \alpha M_\alpha, \tag{27}$$

där α är den varierande parametern och M_0 och M_α är matriser utan något parameterberoende. Om $M(\alpha)$ är diagonaliserbar, enligt:

$$M(\alpha)\phi = E\phi, \tag{28}$$

där ϕ är egenvektorer och E är egenvärden till $M(\alpha)$, spänns underrummet för det minsta egenvärdet, E_{min}, upp av motsvarande egenvektor, ϕ_{min}, för några utvalda värden på α. Med $n < m$ stickprov av α-värden erhålls mängden $\{\phi_i : i = 1, ..., n\}$ stickprov med egenvektorer. Om Φ är $m \times n$ matrisen av stickprov med $m \times 1$ egenvektorer ordnade längs kolonnerna enligt följande:

$$\Phi = \begin{bmatrix} \phi_1 & \phi_2 & \cdots & \phi_n \end{bmatrix}, \tag{29}$$

ges den projicerade $n \times n$ matrisen, $M^*(\alpha)$, av:

$$M^*(\alpha) = \Phi^\top M(\alpha)\Phi. \tag{30}$$

För att bibehålla normen vid beräkning av egenvärden och egenvektorer används $n \times n$ normmatrisen, N, som definieras som följande:

$$N = \Phi^\top \Phi. \tag{31}$$

Den projicerade matrisen, $M^*(\alpha)$, kan då diagonaliseras för att för att erhålla egenvektorer och egenvärden enligt:

$$M^*(\alpha)\phi^* = E^* N \phi^*, \tag{32}$$

där ϕ^* är egenvektorerna och E^* är egenvärdena till $M^*(\alpha)$. Genom RBM har problemet reducerats från att diagonalisera en $m \times m$ matris till att diagonalisera en $n \times n$ matris där precisionen av de emulerade egenvektorerna och egenvärdena beror på valet och antalet stickprov, n, av ϕ. I vissa fall, exempelvis för hamiltonianen, finner man att $E \approx E^*$ med mycket hög noggranhet även för $n << m$. I det här arbetet ska RBM användas för att minska beräkningskostnaderna av att diagonalisera deuteronens hamiltonian, utan större förluster i precisionen av bindningsenergin. Däremot har hamiltonianen ett icke-linjärt beroende av pionmassan som försvårar processen. För att kunna använda RBM måste pionmassans beroende linjäriseras, vilket undersöks genom kontruktionen av en empirisk interpolationsmetod.

2.3.2 Singulärvärdesdekomposition

I detta arbete används singulärvärdesdekomposition (SVD) som ett verktyg för både dimensionsreduktion och som en central komponent i EIM. SVD möjliggör att identifiera de mest informativa riktningarna i datan och därmed konstruera lågdimensionella approximationer av komplexa system.

SVD är en metod för matrisfaktorisering inom linjär algebra. För en godtycklig reell $m \times n$-matris, A, kan SVD skrivas som

$$A = U\Sigma W^T, \tag{33}$$

där U är en $m \times m$ ortogonal matris, W är en $n \times n$ ortogonal matris, och Σ är en $m \times n$ diagonalmatris med icke-negativa singulärvärden $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_{\min(m,n)} \geq 0$ på diagonalen [9]. Kolonnerna i U utgör de ortonormala egenvektorerna av AA^T och spänner upp kolonnrummet till A. Kolonnerna i W är de ortonormala egenvektorerna till $A^T A$ och spänner upp radrummet till A.

Varje singulärt värde är associerat med en basvektor från U och en från W. Större singulärvärden motsvarar riktningar i datan med högre varians, vilket innebär att dessa innehåller mer information [10]. Genom att behålla endast de k största singulärvärdena och deras tillhörande basvektorer erhålls en rang-k-approximation av A som bevarar så mycket av datans struktur som möjligt.

2.3.3 Empirisk Interpolationsmetod

För att hantera icke-linjära beroenden kan EIM (Empiriska Interpolationsmetoden) användas[8]. Den grundläggande idén bakom metoden är att approximera en funktion med icke-linjärt parameterberoende som en summa av termer som är affint beroende av parametern. I detta arbete undersöks det om

man kan approximera det icke-linjära parameterberoendet av m_π i EPU genom EIM och skriva om det på följande form:

$$V_{1\pi}(\vec{p}', \vec{p}) \approx \sum_{q=1}^{k} \beta_q(m_\pi) u_q(\vec{p}', \vec{p}), \tag{34}$$

där $\{u_1, ..., u_k\}$ är basmatriser i Hilbertrummet och $\beta(m_\pi)$ är koefficienter som enbart beror av m_π. Summan i ekvation 34 är den affina expansionen av potentialen $V_{1\pi}(\vec{p}', \vec{p})$.

För att konstruera u_q och β_q används SVD. Antag att potentialen ges som en $N \times N$-matris V, som kan utvärderas för M olika värden på m_π. Detta ger upphov till M olika potentialmatriser, betecknade $V_1, V_2, \ldots, V_M$, där varje V_i motsvarar potentialen för ett specifikt värde $m_\pi^{(i)}$.

Varje sådan matris vektoriseras, vilket innebär att den omformas till en kolumnvektor av storlek $N^2 \times 1$ genom att kolonnerna ordnas i följd. Dessa vektorer samlas i en gemensam matris $\mathcal{V}$ av storlek $N^2 \times M$, definierad som

$$\mathcal{V} = \begin{bmatrix} \text{vec}(V_1) & \text{vec}(V_2) & \cdots & \text{vec}(V_M) \end{bmatrix}, \tag{35}$$

där $\text{vec}(V_i)$ betecknar vektoriseringen av V_i.

Därefter faktoriseras $\mathcal{V}$ med hjälp av SVD enligt

$$\mathcal{V} = U \Sigma W^T. \tag{36}$$

Matrisen U skrivs explicit som

$$U = \begin{bmatrix} u_1 & u_2 & \cdots & u_{N^2} \end{bmatrix}, \tag{37}$$

där $\{u_1, \ldots, u_{N^2}\}$ är ortonormala basvektorer i kolonnrummet till $\mathcal{V}$. För att erhålla en lämplig lågdimensionell approximation väljs endast de k första basvektorerna som svarar mot de största singulärvärdena. Detta ger en ny matris

$$\widetilde{U} = \begin{bmatrix} u_1 & u_2 & \cdots & u_k \end{bmatrix}, \tag{38}$$

som alltså inte beror på m_π, vilket möjliggör en effektiv separation av parameterberoendet.

För att bestämma koefficientfunktionerna $\beta_q(m_\pi)$ löses sedan det linjära ekvationssystemet

$$\widetilde{U} \beta = \mathcal{V}, \tag{39}$$

där β är en $k \times M$-matris av formen

$$\beta = \begin{bmatrix} \beta_1(m_\pi^{(1)}) & \beta_1(m_\pi^{(2)}) & \cdots & \beta_1(m_\pi^{(M)}) \\ \beta_2(m_\pi^{(1)}) & \beta_2(m_\pi^{(2)}) & \cdots & \beta_2(m_\pi^{(M)}) \\ \vdots & \vdots & \ddots & \vdots \\ \beta_k(m_\pi^{(1)}) & \beta_k(m_\pi^{(2)}) & \cdots & \beta_k(m_\pi^{(M)}) \end{bmatrix}, \tag{40}$$

där $\{\beta_1, \ldots, \beta_k\}$ är funktioner som endast beror på m_π.

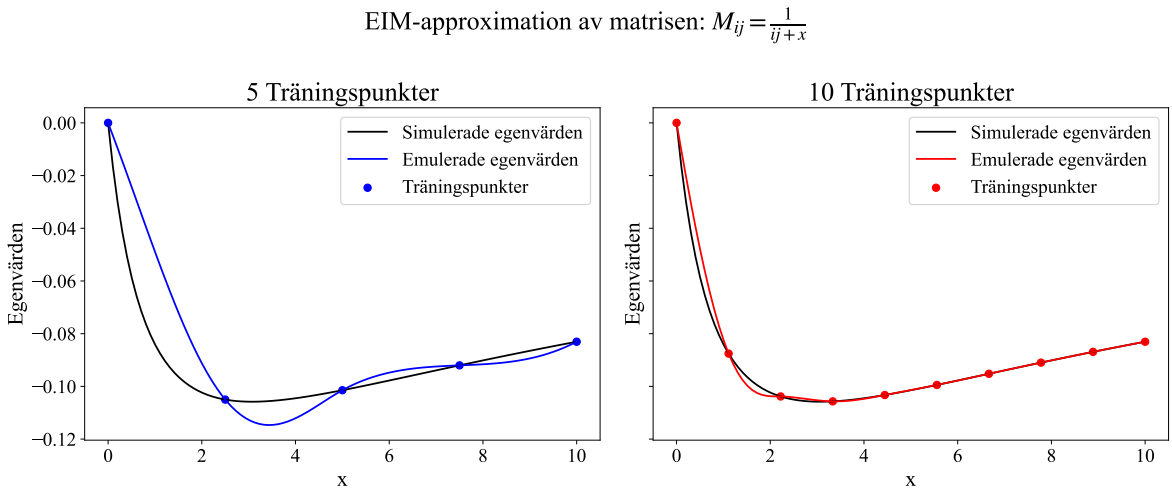
Lösningen till ekvation 39 ger funktionsvärdena för varje β_q i de diskreta punkter $m_\pi^{(1)}, \ldots, m_\pi^{(M)}$. Dessa värden interpoleras därefter för att erhålla kontinuerliga koefficientfunktioner $\beta_q(m_\pi)$ definierade för godtyckliga m_π.

Slutligen återformas de valda basvektorerna i $\widetilde{U}$ till den ursprungliga $N \times N$-matrisstrukturen, vilket fullbordar konstruktionen av den affina approximationen enligt ekvation 34.

Nedanför, i figur 3, hittas ett exempel på en EIM-approximation. En EIM-approximation har gjorts med avseende på parametern x på den godtyckliga matrisen M med matriselement på rad i och kolonn j enligt:

$$M_{ij} = \frac{1}{ij + x}. \tag{41}$$

Figur 3 illustrerar hur ett ökande antal träningspunkter, för värdet på x, har en tydlig korrelation med en ökning av precisionen hos approximeringen, till det lägsta egenvärdet till M.



Figur 3: I figuren syns hur en matris lägsta egenvärden visas som en funktion av variabeln x. Den svarta grafen är de simulerade värdena medan den blåa och röda grafen är en EIM-approximering av matrisen. I den vänstra plotten används 5 stycken träningspunkter och i den högra används 10 stycken.

2.3.4 Kubisk spline-interpolering

För att erhålla kontinuerliga och fysikaliskt rimliga approximationer av koefficientfunktionerna $\beta_q(m_\pi)$ vid interpolering används funktionen CubicSpline från SciPy-biblioteket. Denna funktion utför kubisk spline-interpolering baserat på diskreta funktionsvärden beräknade i ett antal givna m_π-punkter.

En kubisk spline utgörs av en styckvis definierad funktion som är sammansatt av tredjegradspolynom. Varje delpolynom definieras inom ett intervall $[m_\pi^{(i)}, m_\pi^{(i+1)}]$ enligt

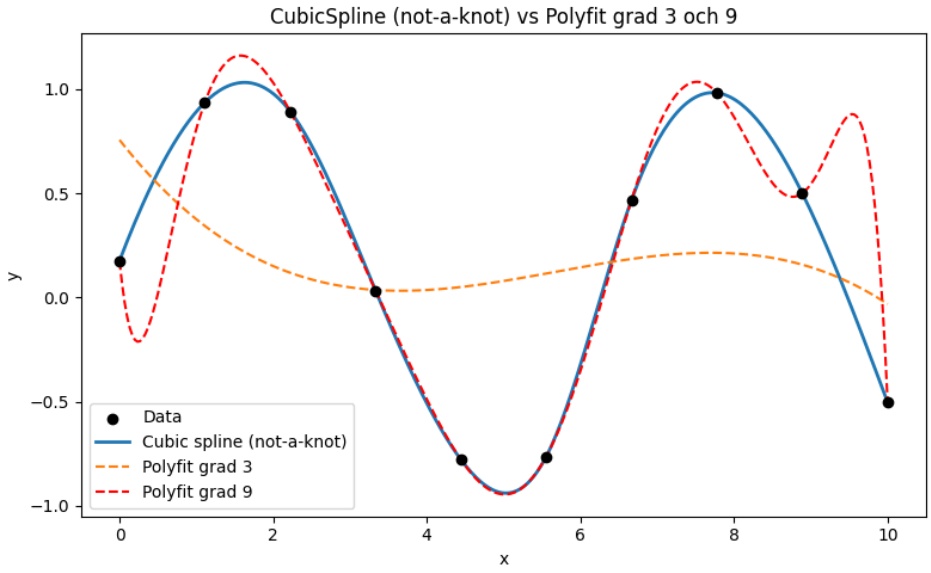
$$\beta_q^{(i)}(m_\pi) = a_i + b_i(m_\pi - m_\pi^{(i)}) + c_i(m_\pi - m_\pi^{(i)})^2 + d_i(m_\pi - m_\pi^{(i)})^3, \quad m_\pi \in [m_\pi^{(i)}, m_\pi^{(i+1)}], \tag{42}$$

och koefficienterna väljs så att funktionen $\beta_q(m_\pi)$ blir kontinuerlig med kontinuerlig första- och andraderivata över hela interpolationsintervallet [11]. För att spline-funktionen ska vara entydigt bestämd krävs att randvillkor specificeras vid ändpunkterna. Vanliga randvillkor inkluderar:

- Naturliga villkor: andra derivatan $\beta_q''(m_\pi)$ sätts till noll vid ändpunkterna.
- Clamped villkor: första derivatan $\beta_q'(m_\pi)$ sätts till noll vid ändpunkterna.
- Periodiska villkor: funktionens och dess derivators värden är lika vid ändpunkterna.
- Not-a-knot villkor: funktion tvingas vara sådan att tredje derivatan är kontinuerlig över de första och sista inre noderna.

Standardinställningen i SciPys CubicSpline innebär att not-a-knot-randvillkor används[12]. Valet av randvillkor påverkar splinens beteende vid ändpunkterna, men i detta arbete läggs ingen särskild vikt vid randvillkoren, eftersom fokus ligger på att erhålla en god interpolation mellan givna datapunkter.

För att ytterliggare motivera valet av kubisk spline-interpolering så visar följande figur en jämförelse mellan kubisk spline-interpolering och global polynomanpassning av olika grad.



Figur 4: Jämförelse mellan kubisk spline-interpolering (not-a-knot) och polynomanpassning med grad 3 och 9 för samma uppsättning diskreta datapunkter. Polynom anpassningen gjordes med NumPys polyfit funktion och kubisk spline-interpoleringen gjordes med CubicSpline från SciPy-biblioteket. Den underliggande datan är en sinus funktion med slumpmässigt brus.

Jämförelsen i figur 4 visar tydligt fördelarna med kubisk spline-interpolering över global polynomanpassning. I figuren så syns det att kubisk spline följer datapunkterna exakt och undviker stora svängningar, medan de globala polynomanpassningarna antingen blir för slätade och missar datapunkterna eller oscillerar kraftigt beroende på gradtal. Detta visar att kubisk spline-interpolering troligtvis är ett lämpligt val för att approximera $\beta_q(m_\pi)$.

2.3.5 Rotmedelkvadratsfel

För en kvantitativ utvärdering av emulatorns prestanda används rotmedelkvadratsfelet (RMSE) som felmått, där emulatorns beräknade värden jämförs med simulerad referensdata.

RMSE är ett mått för att kvantifiera skillnaden mellan observerade och predikterade värden[13]. RMSE definieras enligt:

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}, \tag{43}$$

där N är antalet observationer, y_i är de observerade värdena och $\hat{y}_i$ är motsvarande prediktioner. Ett lågt RMSE indikerar hög överensstämmelse mellan modell och observationer, medan ett högt RMSE signalerar stora genomsnittliga avvikelser.

2.4 Flyttalsoperationer

För att uppskatta beräkningskostnaden för emulatorn används antalet flyttalsoperationer (FLOPs): ett vanligt mått för att kvantifiera mängden numeriskt arbete i en algoritm. En FLOP motsvarar en elementär flyttalsoperation, såsom addition, subtraktion, multiplikation eller division[14].

Med FLOPs för dessa delmoment kan en uppskattning av emulatorns totala beräkningskostnad erhållas, vilket ger en kvantitativ grund för att jämföra effektiviteten med alternativa metoder.

För den gemensamma matrisen $\mathcal{V} \in \mathbb{R}^{N^2 \times M}$ (där $N^2 \geq M$), som uppstår i EIM-steget 2.3.3, krävs följande antal FLOPs för en fullständig SVD [14]:

$$\text{FLOPs: } 4N^4M + 8N^2M^2 + 9M^3.$$

För att lösa det linjära systemet i ekvation 39, vilket används för att bestämma β-funktionerna, är beräkningskostnaden [14]:

$$\text{FLOPs: } 2N^2kM,$$

där k motsvarar antalet basvektorer som behölls för $\widetilde{U}$-matrisen i ekvation 38. Vid kubisk splineinterpolering behandlas varje rad i $\beta \in \mathbb{R}^{k\times M}$ individuellt. En kubisk spline kräver asymptotiskt $\mathcal{O}(M)$ operationer per rad [15], vilket ger en total kostnad:

$$\text{FLOPs: } \sim kM$$

För att extrahera energin från hamiltonoperatorn, samt utföra RBM, krävs diagonalisering av en $n \times n$-matris, vilket har en beräkningskostnad som växer kubiskt med matrisstorleken [14]:

$$\text{FLOPs: } \sim n^3$$

De ovan beräknade momenten utgör de väsentliga operationerna i konstruktionen av emulatorn. Bland dessa är det tydligt att den dominerande kostnaden härrör från singulärvärdesuppdelningen av $\mathcal{V}$, vars komplexitet växer som $\mathcal{O}(N^4M)$. Som jämförelse har en fullständig simulering, där hamiltonoperatorn diagonaliseras separat för varje värde på m_π, en kostnad på $\mathcal{O}(N^3)$ per parameterpunkt.

Antag att bindningsenergin ska utvärderas för L olika värden på m_π. Den totala kostnaden för direkt simulering blir då $\mathcal{O}(LN^3)$, medan emulatorns kostnad domineras av det initiala steget med komplexitet $\mathcal{O}(N^4M)$. För att identifiera när emulatorn blir mer beräkningseffektiv jämförs dessa två uttryck. Om L är tillräckligt stort gäller

$$N^4M < LN^3 \quad \Rightarrow \quad L > NM$$

Detta innebär att emulatorn är mer kostnadseffektiv när antalet utvärderingar L överstiger produkten NM för deuteronen. För ett system med två nukleoner är beräkningskostnaden relativt låg, eftersom hamiltonianen enkelt kan diagonaliseras för $N \approx 100$. I detta fall medför därför emulatorn ingen betydande prestandavinst.

För tyngre kärnor krävs en exponentiellt större bas i Hilbertrummet [16], vilket innebär att dimensionen på hamiltonoperatorn växer som $N \sim e^{cA}$, där c är en positiv konstant och A antalet nukleoner. Detta påverkar dock inte EIM-steget i emulatorn, eftersom det utförs på en två-nukleonpotential med en hanterbar dimension $N \approx 100$. Den ökade beräkningskostnaden uppstår istället i RBM-steget, där $\mathcal{O}(10)$ diagonaliseringar måste göras för hamiltonoperatorn med dimension motsvarande den tyngre kärnan. Eftersom endast ett begränsat antal sådana diagonaliseringar krävs, förväntas emulatorn vara beräkningsmässigt fördelaktig för system med fler än två nukleoner.

3 Konstruktion av Emulatorn

I följande avsnitt presenteras de olika stegen i konstruktionen av emulatorn, som är byggd i Python. För att förenkla användningen av emulatorn så har en klass skapats för de olika stegen. Inom de tidsramarna som tillgavs detta projekt så har klassen utformats för att beräkna deuteronens bindningsenergi som funktion av pionmassan. Klassen är dock modelloberoende och kan hantera emulering av godtycklig ickelinjärt parameterberoende inom matrisdiagonalisering. För att förenkla läsningen kommer detta avsnitt först behandla hur EIM-approximeringen implementeras i koden, sedan om RBM-metoden och till sist ett stycke om prestandan som gjordes av emulatorn vid emulering av deuteronens bindningsenergi. För att kunna köra koden krävs följande Pythonpaket:

- NumPy
- MatPlotLib.PyPlot
- SciPy.LinAlg
- SciPy.Interpolate.CubicSpline
- Bisect

Alla funktioner i klassen kommer inte gås igenom, då de inte är relevanta för just denna rapport, men emulatorns kod samt några extra funktioner, använda för analys går att hitta i appendix A.

3.1 EIM-approximeringen

I detta projekt implementerades EIM för att approximera pionmassans beroende i EPU-potentialen baserat på ett urval av valda interpolationspunkter där pionmassan varierats. Syftet med metoden är att möjliggöra en effektiv, linjär, representation av potentialen, för att därigenom möjliggöra tillämpning av RBM och minska beräkningskostnaden för en godtycklig pionmassa, jämfört med en fullständig simulering.

Implementeringen av EIM genomfördes i Python och följer den algoritmiska struktur som presenteras i avsnitt 2.3.3. En diskret uppsättning av potentialmatriser, utvärderade vid olika värden på m_π, genererades med hjälp av en befintlig kodbas i Python. Urvalet av m_π-värden utfördes genom att definiera ett interpolationsintervall, med NumPy linspace, kring det fysikaliska massvärdet, ett antal värden av pionmassan mellan 0 och 3 gånger det accepterade värdet på pionmassan, $m_\pi = 138.039$ MeV/c_0^2 [17]. Det finns alternativa metoder för val av masspunkter än linspace, vilket diskuteras i avsnitt 5.1.

För att underlätta hanteringen av emulatorn byggdes en klass. Klassens init-funktion hittas nedanför:

```
def __init__(self,matrix,M_pi,min_basis_val=1e-13):
        potential = chiral_potential.two_nucleon_potential('LO',Lambda=500.0)
        dimension=matrix[0].shape[0]
        dim_quadrant=int(dimension/2)
        nn = nn_states.nn_states(
        jmin=0,jmax=1,tzmin=0,tzmax=0,Np=dim_quadrant,mesh_type='gauleg_finite')
        nn.V = potential
        mu = (const.Mp * const.Mn)/(const.Mp + const.Mn)
        pp = np.hstack((nn.pmesh,nn.pmesh))
        Tmat=[p_bra**2/(2*mu) for i, p_bra in enumerate(pp)]
        self.Tmat = np.diag(Tmat)
        self.orgMatrix=matrix
        self.matrix = []
        for i in matrix:
            self.matrix.append(i.flatten())
        self.matrix= np.column_stack(self.matrix)
        self.M_pi = M_pi

        self.beta=[]
```

```
        self.U, self.S, Vt = np.linalg.svd(self.matrix, full_matrices=False)
        basis_val=100*self.S/self.S[0]
        index = bisect.bisect_right(basis_val[::-1], min_basis_val)
        self.n_basisVec = len(basis_val)-index
        self.U=self.U[:,:self.n_basisVec]
        for i in range(0,len(self.M_pi)):
            self.beta.append(np.linalg.lstsq(self.U,self.matrix[:,i] , rcond=None)[0])
        self.beta=np.column_stack(self.beta)

        self.basismatrix=[0]*self.n_basisVec
        for idx,__ in enumerate(self.basismatrix):
            self.basismatrix[idx]=np.array(self.U[:,idx]).reshape(dimension, dimension)

        self.cubicspline = [0]*self.n_basisVec
        for i in range(0,self.n_basisVec):
            self.cubicspline[i]=CubicSpline(self.M_pi, self.beta[i])
        return None
```

Klassen initieras med en mängd variabler. För att enklare förstå vad varje variabel styr kommer här nedanför en lista med förklaring till varje variabel.

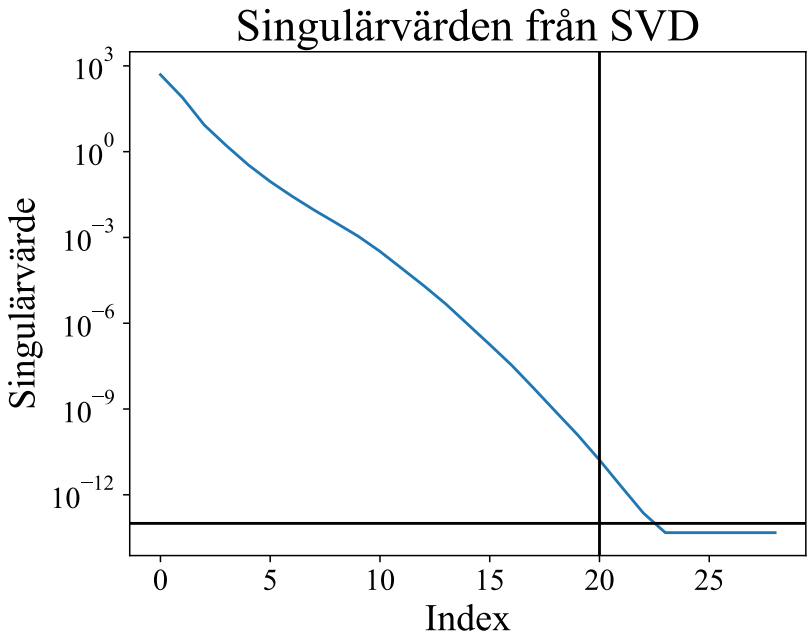
Indata

- matrix: Lista med de matriser (np.array(N,N)) uppbyggda av de valda pionmassorna som ska användas för träningen av EIM-approximeringen.
- M_pi: Lista med de valda träningspunkterna på pionmassan till varje tillhörande matris i **matrix**.
- min_basis_val: Float som används som tröskelvärde för hur många basvektorerna i SVDn som ska användas (se avsnitt 2.3.2). Är satt till 10^{-13} om inget annat anges.

Utdata

- None

Raderna 2 till 11 påverkar inte emulatorns funktion, utan skapar endast det konstanta bidraget från den kinetiska energin till hamiltonianen som nämns i ekvation 10. Mellan raderna 12 till 24 skapas matrisen från ekvation 35 och därefter används NumPys SVD-funktion för att få ut U till EIM-approximationen samt singulärvärdena i Σ-matrisen till varje basvektor (se avsnitt 2.3.2). Därefter används **min_basis_val** som ett tröskelvärde för att ta bort de vektorer med för liten påverkan. Tröskelvärdet som användes var 10^{-13}, vilket togs fram genom en kvantitativ studie där precision och onödigt brus viktades mot varandra. I figur 5 hittas ett exempel på ungefär hur dessa singulärvärden avtar med antalet vektorer i U-matrisen.



Figur 5: I figuren finns en graf över hur Σ-matrisens värden sjunker som funktion av den tillhörande vektorns index. Det är detta värde som **min_basis_val** används som tröskelvärde för.

Till sist beräknas β-koefficienter för varje värde på pionmassan genom ekvation 39 som sedan läggs i en numpy array. Sista delen av koden använder de erhållna β-koefficienterna för att interpolera värdena med Kubisk spline metoden som diskuterades i avsnitt 2.3.4.

3.2 RBM-steget

För att skapa underrumsmatrisen M^*, som beskrivs i ekvation 30, användes följande funktion i klassen:

```
def subspace_basis_RBM(self,RBMmatrix):
        self.Nmatrix_null=False
        sub_basis = []
        M=RBMmatrix
        H=[self.Tmat+V for V in M]
        for i in H:
            w, v = linalg.eigh(i,None)
            s = np.argsort(w)
            v= v[:,s[0]]
            w= w[s[0]]
            norm = np.dot(v.transpose(),v)
            # print('lowest eigenvalue:', w)
            # print('norm <v|v>       :', norm)
            sub_basis.append(v)
        self.sub_basis = np.column_stack(sub_basis)
        self.n_mat = np.transpose(self.sub_basis)@self.sub_basis
        if is_pos_def(self.n_mat)!=True:
            self.Nmatrix_null=True
            print(f' Matrisen har inte positiv definit ({len(M)})')
        self.sub_Mi = [0]*len(self.basismatrix)
        for idx ,M_i in enumerate(self.basismatrix):
            self.sub_Mi[idx]=np.transpose(self.sub_basis).dot(M_i.dot(self.sub_basis))
        self.T_sub=np.dot(
            np.transpose(self.sub_basis),np.dot(self.Tmat,self.sub_basis))
        return None
```

Funktionens in- och utdata hittas i listan nedanför.

Indata

- RBMmatrix: Lista med alla matriser som ska användas för att träna RBM-approximeringen.

Utdata

- None

På raderna 2 till 5 konstrueras hamiltonianen genom att beräkna summan av den kinetiska energimatrisen och potentialmatrisen. Denna procedur är hårdkodad för det specifika fallet av deutronens bindningsenergi och skulle behöva generaliseras för att emulatorn ska kunna tillämpas på andra system. Från rad 5 till 16 sker diagonaliseringen av träningsmatriserna där egenvektorerna för grundtillståndet läggs in i en matris samt att norm-matrisen N, som nämns i ekvation 31, skapas. Därefter kommer en if-sats som avgör om N-matrisen har positiv definit eller inte. Varför detta är viktigt diskuteras mer i avsnitt 4.3. Till sist skapas M^* genom ekvation 30 där både det kinetiska bidraget samt basmatrisen från EIM-approximationen genomgår dessa underrumstransformeringar. Detta leder till att M^* får matrisdimensionerna $M_{RBM} \times M_{RBM}$ där M_{RBM} är antalet träningspunkter i RBM-metoden.

Efter RBM-steget kan underrumsmatriserna användas för att enklare få fram deuteronens bindningsenergi. Detta görs genom följande funktion:

```
1  def evc(self,c_star):
2          Eeim=[0]*len(c_star)
3          if self.Nmatrix_null!=True:
4              for i,m_pion in enumerate(c_star):
5                  H_sub=self.T_sub.copy()
6                  for idx, mat in enumerate(self.sub_Mi):
7                      H_sub+=mat*self.cubicspline[idx](m_pion)
8                  eigvals, eigvecs = linalg.eigh(H_sub,self.n_mat)
9                  s = np.argsort(eigvals)
10                 Eeim[i] = eigvals[s[0]]
11                 psi_k = eigvecs[:,s[0]]
12             return Eeim
13         else:
14             return None
```

Indata och utdata hittas i listan nedanför.

Indata

- c_star: Lista med alla pionmassor som ska emuleras.

Utdata

- Eeim: Lista med bindningenergin för pionmassorna i c_star.

Anledningen till if-satsen som finns med i funktionen kommer att diskuteras mer i avsnitt 4.3. Inuti if-satsen adderas först underrumsmatrisen för den kinetiska energin till **H_sub**. Därefter adderas produkten av varje beta-koefficient och dess motsvarande underrumsmatris för potentialen till samma matris. När alla matriser har adderats så har den slutgiltiga hamiltonianen i underrummet skapats, **H_sub**. Denna matris diagonaliseras sedan, varifrån de emulerade bindningsenergierna extraheras från de lägsta egenvärdena och returneras i en lista.

4 Resultat av Emulatorns karaktäristik och felanalys

I detta avsnitt kommer de resultat av emulatorns karaktäristik och den felanalys som genomförts att presenteras. Den primära feldiagnostiken som har gjort är baserad på RMSE (Rotmedelkvadratsfel) som förklaras i avsnitt 2.3.5. Därefter presenteras resultaten för EIM- och RBM-approximeringen för att sedan avslutas med en övergripande felanalys av hela emulatorn. Alla potentialmatriser har dimensionerna 140×140 med ett $\Lambda = 500$ MeV, om inget annat anges. Λ-parametern kommer inte diskuteras mer i detta arbete utan förklaras kort i avsnitt 2.1.2.

Vissa resultat kommer att jämföras mot den experimentellt uppmätta mätfelet av bindningsenergin av en deuteron som under arbetets gång befinner sig på $\pm 2.2 \cdot 10^{-6}$ MeV/c_0^2 [18]. De pionmassorna som användes för att träna både EIM-approximationen och RBM-metoden var alltid jämnt utspridda punkter mellan 0 MeV/c_0^2 och $3m_\pi$, och emuleringsdatan som användes för att evaluera modellen för pionmassan var 1000 stycken, jämnt utspridda, punkter på samma intervall.

4.1 Felanalys

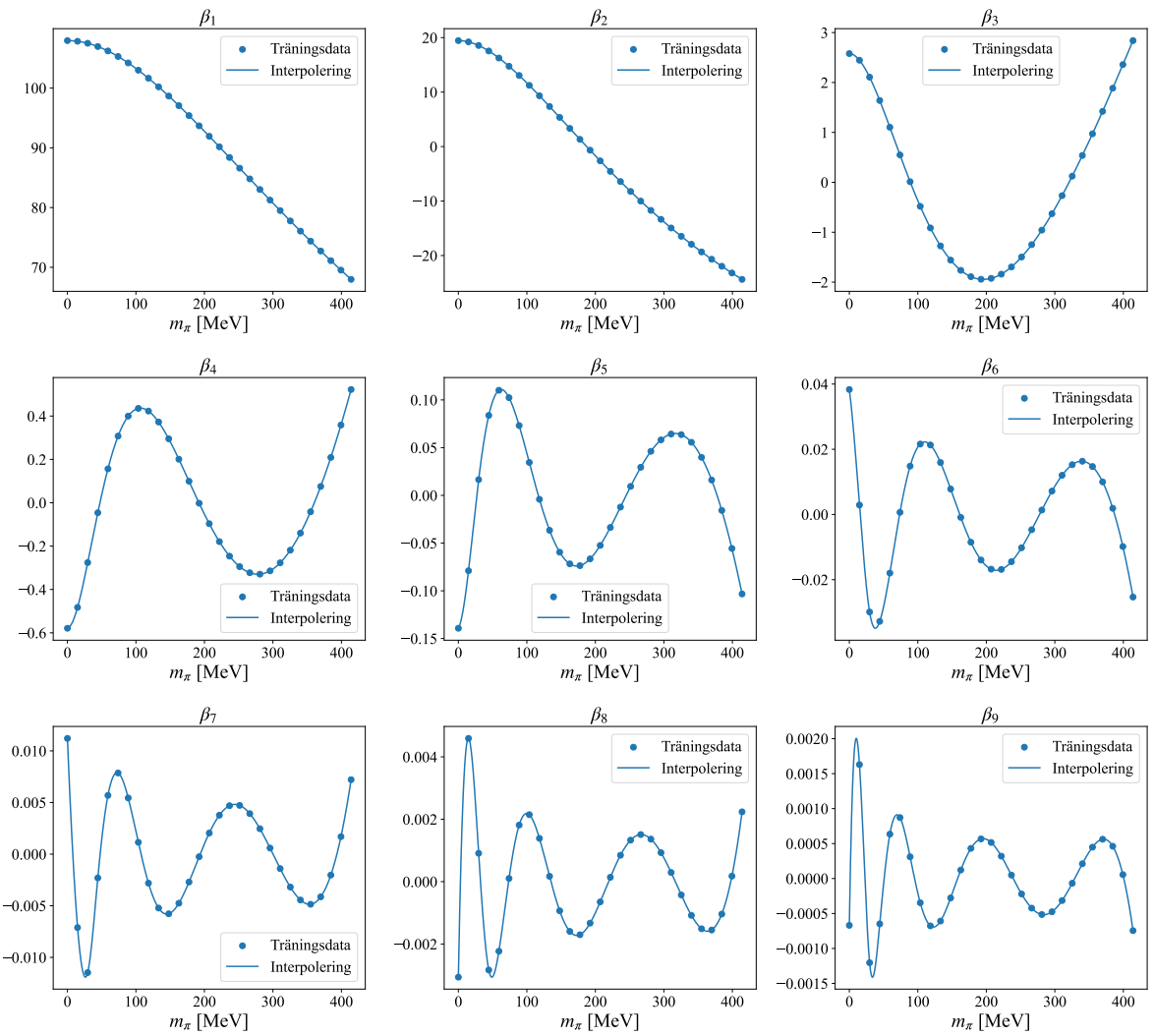
För att utvärdera hur väl EIM och RBM presterade användes RMSE som felmått. Båda metoderna jämfördes mot separat valideringsdata i flera steg för att kvantifiera var eventuella felkällor uppstår. Den simulerade datan konstruerades genom att diagonalisera 1000 stycken hamiltonianer med en pionmassa som varierade mellan 0 och $3m_\pi$, för att beräkna bindningsenergin över hela det interpolationsintervallet som kommer användas i senare resultat. Detta resulterade i en noggrann och kontinuerlig beskrivning av deuteronens bindningsenergi som funktion av pionmassan. Den koden som användes för att beräkna RMSE-värdet för olika punkter var följande:

```python
def calculate_rmse(true_x, true_y, model_x, model_y):

    true_spline = CubicSpline(true_x, true_y)

    true_y_interp = true_spline(model_x)

    rmse = np.sqrt(np.mean((true_y_interp - model_y) ** 2))
    return rmse
```

Eftersom datorer jobbar med diskreta punkter används kubisk spline för att interpolera de simulerade värdena så att vi får ett kontinuerligt spektrum av simulerade punkter. Istället för att ta det närmsta emuleringsvärdet och jämföra med det emulerade så interpoleras de simulerade punkterna emellan så en kontinuerlig funktion erhölls, vilket bidrar till en större precision på analysen. Kodens funktion fungerar precis som beskriven i ekvation 43.

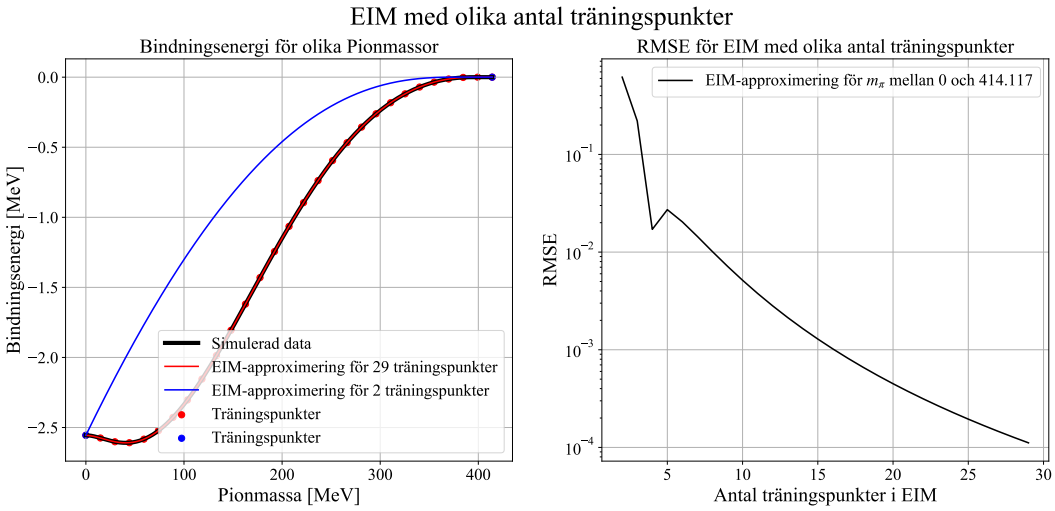
4.2 Karaktäristik och feldiagnostik av EIM-approximeringen

I figur 6 syns de β-koefficienterna som är kopplade till de nio stycken största singulärvärdena som funktion av pionmassan. De inledande funktionerna uppvisar en polynomiell struktur, och från koefficient nummer 2 ökar graden med ett för varje efterföljande funktion. Det är också tydligt från figuren att β-koefficienternas värde minskar med ungefär en halv 10-potens för varje nytt β.



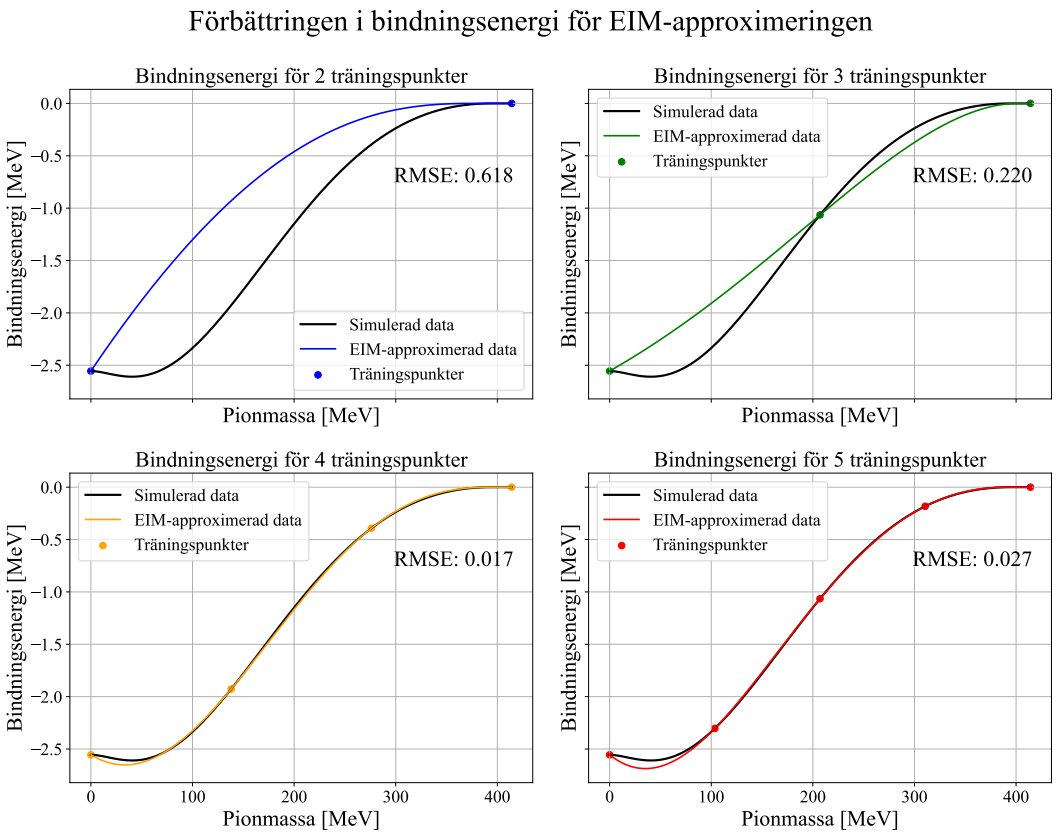
Figur 6: I figuren syns de nio stycken första β-koefficienterna i EIM-approximeringen som funktion av pionmassan. De är tydligt att de första (och största) koefficienterna har en mindre rörelse än de senare som rör sig mer oförutsägbart men som i sin tur har en lägre påverkan. Denna trend fortsätter även för högre index av β.

Feluppskattningen av EIM-approximationens påverkan på bindningsenergin visas i figur 7. I den vänstra panelen visas bindningsenergin som funktion av pionmassan för emulatorer med olika antal träningspunkter. Den svarta kurvan representerar den simulerade bindningsenergin, medan de färgade linjerna visar de EIM-approximerade bindningsenergierna där 2 respektive 29 punkter har använts i träningen. I den högra panelen illustreras hur RMSE-värdet avtar med ökande antal träningspunkter i EIM-approximationen. Viktigt att poängtera är att ingen RBM-metod har använts i denna analys utan istället har de EIM-approximerade matriserna diagonaliserats där det lägsta egenvärdet representerar bindningsenergin. Felet minskar snabbt redan efter 5 träningspunkter, och fortsätter minska därefter, med avtagande förbättring vid fler träningspunkter. För 29 träningspunkter erhålls ett RMSE på cirka 1.1×10^{-4}. Detta värde är relativt långt ifrån osäkerheten i den experimentellt uppmätta pionmassan som nämndes i avsnitt 4, men ses ändå som ett fördelaktigt resultat då deuteronens bindningsenergi, som funktion av pionmassan, är rapportens fokus.



Figur 7: I figuren visas två diagram där det vänstra presenterar tre kurvor som återger bindningsenergin som funktion av pionmassan; den blå kurvan motsvarar en EIM-approximation med två träningspunkter, den röda en EIM-approximation med 29 träningspunkter, och den svarta kurvan representerar simulerad data. I den högra grafen syns RMSE-värdet av EIM-approximeringen som funktion av antalet träningspunkter i EIM-approximationen.

För att tydligare illustrera hur antalet träningspunkter påverkar EIM-approximationen, presenteras figur 8. Figuren visar de fyra första träningsstegen för EIM-approximationen som tidigare redovisats i figur 7. Det framgår att approximationen med enbart två träningspunkter ger ett begränsat resultat, men att felet snabbt minskar när ytterligare träningspunkter inkluderas. Däremot ökar felet från den 3:e till fjärde figuren. Detta är troligen på grund av den skarpa förändringen av bindningsenergin när pionmassan närmar sig 0. Detta tyder på att fler träningspunkter i intervallet 0 MeV/$c_0^2 < m_\pi <$ 100 MeV/c_0^2 hade varit mer kostnadseffektivt än att lägga till en extra punkt i Numpys linspace-funktion.

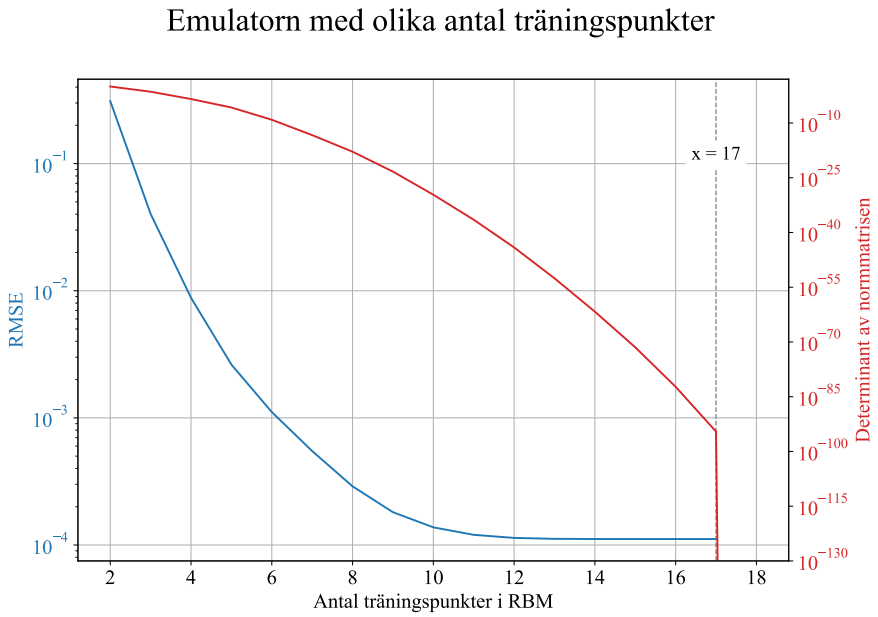


Figur 8: I figuren syns hur de fyra första träningspunkterna från figur 7 påverkar den slutliga interpolerade bindningsenergin.

Samtidigt är det värt att notera att även med ett litet antal träningspunkter, exempelvis 5 st, fås en god approximation av den övergripande trenden i hur pionmassan påverkar bindningsenergin. Detta antyder att metoden har potential att användas även i större system, där det inte är praktiskt möjligt att använda ett stort antal träningsdata.

4.3 Karaktäristik och felanalys av RBM-approximeringen

I figur 9 visas RMSE-värdet från RBM-steget som funktion av antalet träningspunkter som använts för att konstruera basen, enligt beskrivningen i avsnitt 34. Eftersom RBM-metoden inte kan hantera det ickelinjära beroendet av pionmassan direkt, används en EIM-approximation baserad på 29 träningspunkter för att möjliggöra linjärisering.



Figur 9: Figuren visar ett diagram med två stycken grafer, där den blåa visar RMSE-värdet av Emulatorn, som får en asymptot vid ungefär 1.1×10^{-4}. Den röda grafen visar determinanten av normmatrisen, N, som nämns i ekvation 31, och ett tydligt avbrott som syns vid 17 träningspunkter, illustrerad av linjen $x = 17$. Vid detta värde får N-matrisen en negativ definit och determinanten blir därför negativ.

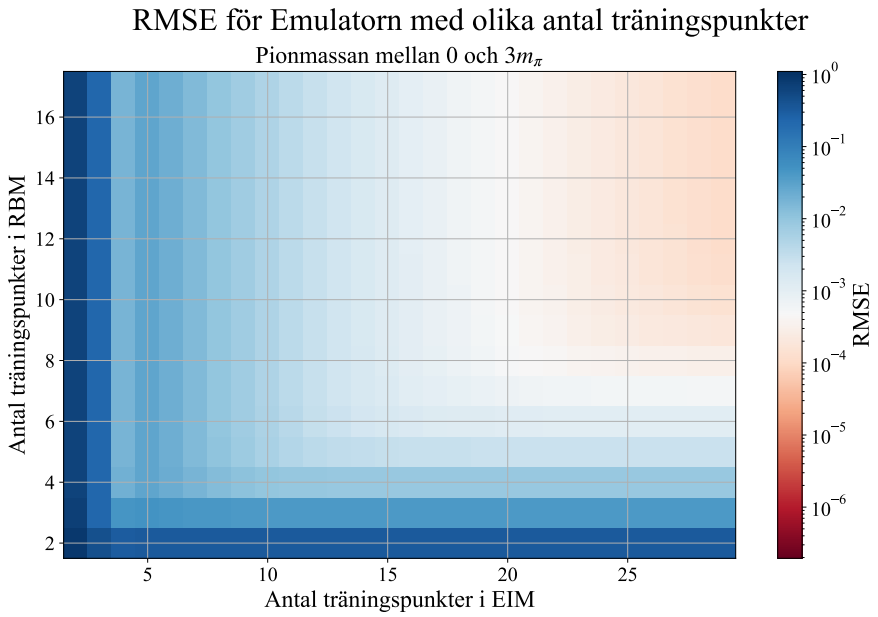
Resultaten visar att RMSE minskar kontinuerligt med antalet basvektorer, men förbättringen planar ut kring 15 st träningspunkter. Utöver detta visas även determinanten av normmatrisen N, som ingår i beräkningen av egenvärden. Denna determinant minskar snabbt och blir numeriskt negativ när fler än 17 träningspunkter används. Detta indikerar att normmatrisen blir numeriskt singulär, vilket sannolikt beror på att antalet basvektorer i Φ (se ekvation 29) blir för stort. Då riskerar flera vektorer att bli linjärt beroende inom det genererade underrummet. Detta innebär att ytterligare träningspunkter inte förbättrar emuleringen, vilket bekräftas av att RMSE-värdet inte sjunker nämnvärt efter 15 basvektorer. Valet av antalet baser är därmed en balans mellan noggrannhet och numerisk stabilitet.

4.4 Felanalys av fullständiga emulatorn

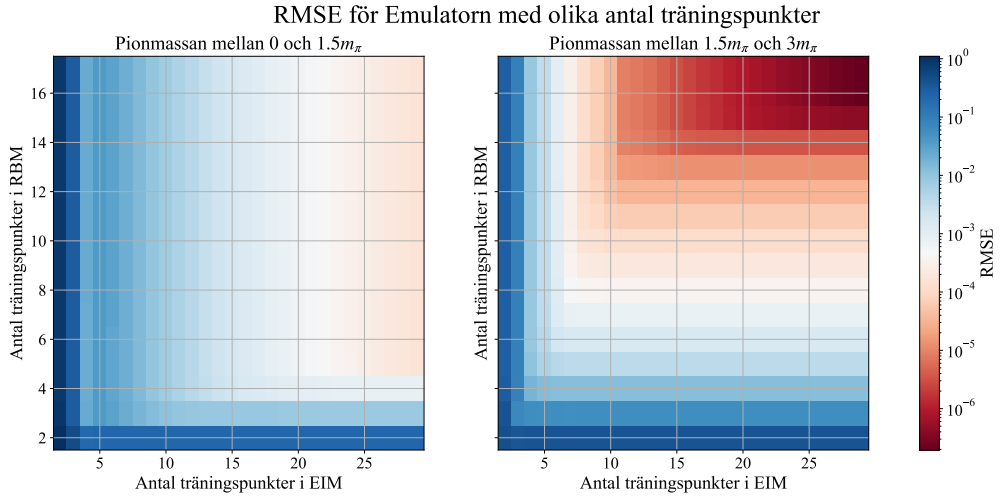
I figur 10 och 11 illustreras RMSE-värdet som funktion av antalet träningspunkter i EIM-approximationen respektive RBM-metoden. Figur 10 visar RMSE för ett intervall av pionmassor mellan 0 och $3m_\pi$, där felet avtar relativt jämnt med ökat antal träningspunkter, och når ett minimumvärde runt 1.1×10^-4.

Det är dock viktigt att notera att antalet träningspunkter i RBM-steget inte kan ökas utan att emulatorn slutar att fungera, när orginalmatrisernas dimension ligger på 70. Detta beror på att dimensionen hos hamiltonianen sätter en övre gräns för antalet linjärt oberoende basvektorer. När denna gräns överskrids leder detta till att N-matrisens determinant blir negativ, som diskuterats i avsnitt 4.3. Ytterligare ökning av träningspunkter leder därför inte till förbättrad precision.

I figur 11 visas RMSE separat för två olika delintervall: det vänstra diagrammet avser intervallet (0) till $1.5m_\pi$, medan det högra avser $1.5m_\pi$ till $3m_\pi$. Resultaten visar tydligt att EIM-approximationen presterar sämre i det lägre massintervallet. Detta är sannolikt kopplat till att den största variationen i bindningsenergin sker just där, vilket också bekräftas av de simulerade resultaten i figur 8. För att förbättra noggrannheten i emuleringen i det lägre intervallet bör därför fokus ligga på att förtäta träningspunkterna i EIM-approximationen. I det högre intervallet, där variationen är mindre och EIM presterar bättre, är det RBM-metoden som kräver flest träningspunkter för att se en förbättring i RMSE-värdet.



Figur 10: Figuren visar en värmekarta för emulatorn, där antal träningspunkter i EIM-approximationen finns på x-axeln, antalet träningspunkter i RBM-approximationen finns på y-axeln och färgen representerar RMSE-värdet. I grafen är bindingsenergin emulerad med en pionmassa mellan 0 till $3m_\pi$.



Figur 11: Figuren visar två värmekartor för emulatorn, där antal träningspunkter i EIM-approximationen finns på x-axeln, antalet träningspunkter i RBM-approximationen finns på y-axeln och färgen representerar RMSE-värdet. I den vänstra grafen är bindingsenergin emulerad med en pionmassa mellan 0 till $1.5m_\pi$ och den högra grafen mellan $1.5m_\pi$ till $3m_\pi$.

5 Diskussion

Syftet med detta arbete var att konstruera en effektiv emulator för att approximera bindningsenergin hos deuteronen genom att skapa en affin approximation av EPU potentialen med EIM, följt av tillämpning av RBM. Den resulterande modellen visade mycket god prestanda redan vid ett litet antal träningspunkter. Framför allt gav fem eller fler träningspunkter noggranna approximationer, vilket visar att EIM metoden lyckas fånga potentialens parameterberoende med relativt låg beräkningskostnad.

5.1 Övergripande diskussion

Resultaten indikerar att EIM metoden framgångsrikt fångar deuteronens icke-linjära beroende av m_π i potentialen trots kraftig dimensionsreduktion. En begränsning i studien är att emulatorn utvärderats enbart för deuteronen och EPU-potentialen i ett fastlagt parameterintervall. Det är ännu oklart hur modellen skulle prestera för mer komplexa pion-interaktioner och kärnor med fler nukleoner. Som konsekvens av detta kan inget konkret sägas om emulatorn generaliserbarhet för mer komplexa system.

I detta arbete valdes träningspunkterna med ett jämnt avstånd över parameterintervallet genom NumPys funktion linspace. Denna metod är enkel att implementera och gav goda resultat i detta fall, men säkerhetsställer inte att optimala punkter väljs ut. Eftersom vissa delar av parameterområdet kan bidra mer till systemets varians än andra, kan ett mer optimalt urval förbättra effektiviteten ytterligare. Det pekar mot att valet av träningspunkter är en viktig komponent i emulatorns prestanda och bör studeras mer systematiskt.

Användningen av EIM för att konstruera en affint separerad approximation av potentialen innan tillämpningen av RBM visade sig vara en effektiv strategi. Genom att linjärisera problemet i ett förberedande steg reduceras den numeriska komplexiteten jämfört med att direkt hantera en icke-linjär operator. Samtidigt bevaras viktiga strukturella egenskaper hos modellen. Metoden illustrerar hur väl utformade numeriska tekniker kan kombineras för att uppnå både hög noggrannhet och beräkningseffektivitet vid emulering av kvantmekaniska system.

5.2 Slutsats

Arbetet visar att det är möjligt att med hög noggrannhet emulera deuteronens bindningsenergi genom att kombinera EIM och RBM. Resultaten pekar på att metoden är beräkningsmässigt effektiv och erbjuder ett systematiskt sätt att reducera dimensionen hos problemet utan att förlora viktig fysikalisk information. Metoden kan därför ses som ett första steg mot mer avancerade emulatormodeller för kvantmekaniska system, och resultaten bekräftar att kombinationen av EIM och RBM är väl lämpad för denna typ av fysikaliska problem.

5.3 Fortsatt arbete

För att vidare utveckla metoden finns det flera intressanta riktningar att undersöka. En viktig fråga är hur väl emulatorn presterar för mer komplexa växelverkansmodeller och, i synnerhet, för kärnor med fler nukleoner. För sådana system växer den beräkningsmässiga kostnaden snabbt och gör direkta lösningar för parameterstudier praktiskt ogenomförbara, vilket ytterligare motiverar behovet av effektiva emulatormetoder. En utvärdering i dessa mer realistiska och beräkningstunga tillämpningar skulle ge en djupare förståelse för metodens generaliserbarhet och praktiska användbarhet.

Vidare vore det av stort värde att undersöka mer avancerade strategier för val av träningspunkter. I detta arbete användes ett jämnt punkturval över parameterintervallet, men en girig algoritm som aktivt söker de områden som bidrar mest till variansen skulle sannolikt kunna förbättra konvergensen ytterligare. Att använda sådana informationsdrivna urval skulle göra emulatorn mer effektiv, särskilt i fall där möjligheten att utvärdera potentialen för fler träningspunkter är begränsad.

Referenser

[1] E. Rutherford. "The scattering of α and β particles by matter and the structure of the atom." I: *Philosophical Magazine Series 6* 21.125 (1911), s. 669–688. DOI: 10.1080/14786440508637080.

[2] Hideki Yukawa. "On the Interaction of Elementary Particles. I". I: *Proceedings of the Physico-Mathematical Society of Japan* 17 (1935), s. 48–57.

[3] R. Machleidt och D.R. Entem. "Chiral effective field theory and nuclear forces". I: *Physics Reports* 503.1 (juni 2011), 1–75. ISSN: 0370-1573. DOI: 10.1016/j.physrep.2011.02.001. URL: http://dx.doi.org/10.1016/j.physrep.2011.02.001.

[4] David J Griffiths och Darrell F Schroeter. *Introduction to quantum mechanics*. Cambridge university press, 2019.

[5] S. S. M. Wong. *Introductory nuclear physics*. 1998.

[6] R. H. Landau, M. J. Páez och C. C. Bordeianu. "Quantum Bound States via Integral Equations". I: *Computational Physics: Problem Solving with Computers*. Weinheim, Germany: Wiley-VCH, 2007. Kap. 29, s. 443–447.

[7] Andreas Ekström. *Bayesian inference and modelling of nuclear forces*. Presentation. Presenterad 2023-07-10.

[8] Jan S Hesthaven, Gianluigi Rozza, Benjamin Stamm m. fl. *Certified reduced basis methods for parametrized partial differential equations*. Vol. 590. Springer, 2016.

[9] V. Klema och A. Laub. "The singular value decomposition: Its computation and some applications". I: *IEEE Transactions on Automatic Control* 25.2 (1980), s. 164–176. DOI: 10.1109/TAC.1980.1102314.

[10] Trevor Hastie. *The elements of statistical learning: data mining, inference, and prediction*. 2009.

[11] C. De Boor. *A Practical Guide to Splines*. Vol. 27. Applied Mathematical Sciences. New York: Springer-Verlag, 1978.

[12] SciPy Developers. *SciPy documentation: CubicSpline*. https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.CubicSpline.html. Hämtad: 2025-04-18. 2025.

[13] Cort J Willmott och Kenji Matsuura. "Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance". I: *Climate research* 30.1 (2005), s. 79–82.

[14] Gene H Golub och Charles F Van Loan. *Matrix computations*. JHU press, 2013.

[15] Steven Punshon-Smith. *Lecture Notes on Splines*. https://math.tulane.edu/~spunshonsmith/amsc460spring18/files/splines.pdf. Accessed: 2025-05-06. 2018.

[16] Axel Pérez-Obiol m. fl. "Nuclear shell-model simulation in digital quantum computers". I: *Scientific Reports* 13.1 (2023), s. 12291.

[17] C. Nordling och J. Österman. *Physics Handbook: For Science and Engineering*. MTM, 2023. ISBN: 9789130080960. URL: https://books.google.se/books?id=4J800AEACAAJ.

[18] A.H. Wapstra. "Energy calibration for 2–13 MeV gamma rays". I: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 292.3 (1990), s. 671–676. ISSN: 0168-9002. DOI: https://doi.org/10.1016/0168-9002(90)90185-9. URL: https://www.sciencedirect.com/science/article/pii/0168900290901859.

A Emulatorns kod

Nedanför finns den kod som användes till emulatron samt lite andra funktioner som användes vid analysen av Emulatorn. Följande är också en permanent länk till hela kodbibloteket: `https://gitlab.com/andreas-ekstrom/bsc_2025_code/-/tree/f4515ad9a4eb2888a1beabbcfa25b585809f28b3/`.

```
class Emulator_NN:
    def __init__(self,matrix,M_pi,min_basis_val=1e-13):
        potential = chiral_potential.two_nucleon_potential('LO',Lambda=500.0)
        dimension=matrix[0].shape[0]
        dim_quadrant=int(dimension/2)
        nn = nn_states.nn_states(jmin=0,jmax=1,tzmin=0,tzmax=0,Np=dim_quadrant,mesh_type='gauleg_finite')
        nn.V = potential
        mu = (const.Mp * const.Mn)/(const.Mp + const.Mn)
        pp = np.hstack((nn.pmesh,nn.pmesh))
        Tmat=[p_bra**2/(2*mu) for i, p_bra in enumerate(pp)]
        self.Tmat = np.diag(Tmat)
        self.orgMatrix=matrix
        self.matrix = []
        for i in matrix:
            self.matrix.append(i.flatten())
        self.matrix= np.column_stack(self.matrix)
        self.M_pi = M_pi

        self.beta=[]
        self.U, self.S, Vt = np.linalg.svd(self.matrix, full_matrices=False)
        basis_val=100*self.S/self.S[0]
        index = bisect.bisect_right(basis_val[::-1], min_basis_val)
        self.n_basisVec = len(basis_val)-index
        self.U=self.U[:,:self.n_basisVec]
        for i in range(0,len(self.M_pi)):
            self.beta.append(np.linalg.lstsq(self.U,self.matrix[:,i] , rcond=None)[0])
        self.beta=np.column_stack(self.beta)

        self.basismatrix=[0]*self.n_basisVec
        for idx,__ in enumerate(self.basismatrix):
            self.basismatrix[idx]=np.array(self.U[:,idx]).reshape(dimension, dimension)

        self.cubicspline = [0]*self.n_basisVec
        for i in range(0,self.n_basisVec):
            self.cubicspline[i]=CubicSpline(self.M_pi, self.beta[i])
        return None

    def EIMmatrix(self,Mpi):
        cubic_spline_matrix = len(Mpi)*[0]
        for i in range(0,len(Mpi)):
            summatrix=np.zeros((np.shape(self.basismatrix[0])))
            for j ,func in enumerate(self.cubicspline):
                summatrix+= self.basismatrix[j]*func(Mpi[i])
            cubic_spline_matrix[i] = summatrix
        return cubic_spline_matrix


    def subspace_basis_EIM(self):
        plt.title("Singulärvärden från SVD")
        plt.semilogy(self.S)
        plt.xlabel("Index")
        plt.axhline(1e-13,c='black')
        plt.axvline(23,c='black')
        plt.text(8, 1e-12, r'Singulärvärde = $10^{-13}$', rotation=0, ha='center', va='bottom', size=15)
        plt.text(18, 1, 'Index = 23', rotation=0, ha='center', va='bottom', size=15)
```

```
        print(self.S[-1])
        plt.ylabel("Singulärvärde")
        plt.show()
        plt.title("Variance of V_matrix first column")
        plt.plot(np.var(self.matrix.T,axis=0))
        plt.show()
        return None

    def singular_values_for_subspace(self):
        for i in range(self.n_basisVec):
            plt.plot(self.U[:,i], label="sing. val. ratio {:1.3f}%".format(100*self.S[i]/self.S[0]))
        plt.title("First {} U-vectors vectors".format(self.n_basisVec+1))
        plt.xlabel("Basis vector index")
        plt.ylabel("U-vector value")
        plt.legend()
        plt.show()
        return None

    def plot_beta_on_mpi(self):
        MpiSmooth=np.linspace(min(self.M_pi),max(self.M_pi),1000)
        fig,ax=plt.subplots(3,3,figsize=(17,16))
        num=0
        for i in range(0,3):
            for j in range(0,3):
                ax[i][j].grid()
                ax[i][j].scatter(self.M_pi,self.beta[num],label="Träningsdata")
                ax[i][j].plot(MpiSmooth,self.cubicspline[num](MpiSmooth),label="Interpolerad funktion")
                ax[i][j].set_title(r"$\beta$-koefficient nr {} ".format(num+1),size=20)
                ax[i][j].set_xlabel(r"$m_\pi$ [MeV]",size=20)
                ax[i][j].set_ylabel(r"Värde av $\beta$-koeefficient",size=20)
                ax[i][j].legend()
                num+=1
        fig.suptitle(r"De 9 st första $\beta$-koefficienterna",size=30)
        plt.tight_layout()
        plt.show()
        return None

    def error_analysis(self):
        self.Felmatrix=self.U@self.beta-self.matrix
        print("Felmatrisen:", self.Felmatrix)
        print("Maxvärde felmatris",np.max(self.Felmatrix))
        print("Minvärde matris",np.min(self.matrix))
        print("Maxvärde matris", np.max(self.matrix))
        return None
    # nn = nn_states.nn_states(jmin=0,jmax=1,tzmin=0,tzmax=0,Np=70,mesh_type='gauleg_finite')
    # mu = (const.Mp * const.Mn)/(const.Mp + const.Mn)

    def Ebindnin(self,V):
        Eeim=[0]*len(V)
        self.Hlist=[]
        for idx,m in enumerate(V):
            H = self.Tmat+m
            self.Hlist.append(H)
            eigvals, eigvecs = linalg.eigh(H)
            s = np.argsort(eigvals)
            Eeim[idx] = eigvals[s[0]]
            psi_k = eigvecs[:,s[0]]
        return Eeim


    def subspace_basis_RBM(self,RBMmatrix):
```

```
        self.Nmatrix_null=False
        sub_basis = []
        M=RBMmatrix
        H=[self.Tmat+V for V in M]
        for i in H:
            w, v = linalg.eigh(i,None)
            s = np.argsort(w)
            v= v[:,s[0]]
            w= w[s[0]]
            norm = np.dot(v.transpose(),v)
            # print('lowest eigenvalue:', w)
            # print('norm <v|v>       :', norm)
            sub_basis.append(v)
        self.sub_basis = np.column_stack(sub_basis)
        self.n_mat = np.transpose(self.sub_basis)@self.sub_basis
        if is_pos_def(self.n_mat)!=True:
            self.Nmatrix_null=True
            print(f' Matrisen har inte positiv definit ({len(M)})')
        self.sub_Mi = [0]*len(self.basismatrix)
        for idx ,M_i in enumerate(self.basismatrix):
            self.sub_Mi[idx]=np.transpose(self.sub_basis).dot(M_i.dot(self.sub_basis))
        self.T_sub=np.dot(
            np.transpose(self.sub_basis),np.dot(self.Tmat,self.sub_basis))
        return None

    def evc(self,c_star):
        Eeim=[0]*len(c_star)
        if self.Nmatrix_null!=True:
            for i,m_pion in enumerate(c_star):
                H_sub=self.T_sub.copy()
                for idx, mat in enumerate(self.sub_Mi):
                    H_sub+=mat*self.cubicspline[idx](m_pion)
                eigvals, eigvecs = linalg.eigh(H_sub,self.n_mat)
                s = np.argsort(eigvals)
                Eeim[i] = eigvals[s[0]]
                psi_k = eigvecs[:,s[0]]
            return Eeim
        else:
            return None
def E_bind_real(Mpi,dim=70,Lambda=500):
    Eeim=[0]*len(Mpi)
    V=potential_temp(dim,Lambda=Lambda,M_pi=Mpi)
    Hlist=[]
    nn_star = nn_states.nn_states(jmin=0,jmax=1,tzmin=0,tzmax=0,Np=dim,mesh_type='gauleg_finite')
    mu = (const.Mp * const.Mn)/(const.Mp + const.Mn)
    pp = np.hstack((nn_star.pmesh,nn_star.pmesh))
    Tmat=[p_bra**2/(2*mu) for i, p_bra in enumerate(pp)]
    Tmat = np.diag(Tmat)
    for idx,m in enumerate(V):
        H = Tmat+m
        Hlist.append(H)
        eigvals, eigvecs = linalg.eigh(H)
        s = np.argsort(eigvals)
        Eeim[idx] = eigvals[s[0]]
        psi_k = eigvecs[:,s[0]]
    return Eeim

def calculate_rmse(true_x, true_y, model_x, model_y):

    true_spline = CubicSpline(true_x, true_y)

```

```
    true_y_interp = true_spline(model_x)

    rmse = np.sqrt(np.mean((true_y_interp - model_y) ** 2))
    return rmse
def is_pos_def(A):
    try:
        np.linalg.cholesky(A)
        return True
    except np.linalg.LinAlgError:
        return False
```

INSTITUTIONEN FÖR FYSIK
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2025
www.chalmers.se

CHALMERS