CHALMERS
UNIVERSITY OF TECHNOLOGY

Datoruppgift – en realistisk vågfunktion för deuteronen

Jonatan Haraldsson jonhara@chalmers.se
Oscar Lindberg oscarlin@chalmers.se

Program: Teknisk fysik.

Kurs: Subatomär fysik, FUF050.

April, 2023

1 Inledning

Deuteronen, kärnan till väteisotopen deuterium ($^2_1\mathrm{H}_1$), är ett kvantmekaniskt system som består av en proton (p^+) och en neutron (n) bundna av stark växelverkan. Matematiskt beskrivs systemet av Schrödingerekvationen, som ofta löses numeriskt. I denna rapport presenteras teori, metod och resultat då Numerovs numeriska metod användes för att lösa Schrödingerekvationen för deuteronen med en känd Malfliet-Tjon-potential. Från resultatet beräknades deuteronens RMS-radie och bindningsenergi.

2 Teori

Dynamiken i ett kvantmekaniskt system beskrivs av Schrödingerekvationen

$$\hat{H}\Psi(\vec{r}) = (\hat{T} + \hat{V})\Psi(\vec{r}) = E\Psi(\vec{r}),$$

där $\hat{T}$ och $\hat{V}$ motsvarar kinetisk respektive potentiell energi, E är systemets energi och $\Psi(\vec{r})$ dess vågfunktion. Potentialen för deuteronen, den så kallade Malfliet-Tjon-potentialen, ges av

$$V_{\mathrm{MT}}(r) = \sum_{i=1}^{3} V_i \frac{e^{-a_i r}}{r}, \tag{1}$$

där V_i och a_i är konstanter. Då Malfliet-Tjon potentialen är sfäriskt symmetrisk och vågfunktionen för deuteronen en superposition av två ℓ-tillstånd, $\ell = 0, 2$ där $\ell = 2$ försummas, räcker det att lösa den endimensionella Schrödingerekvationen i den polära radien r. Schrödingerekvationen är en andra ordningens differentialekvation, vilken efter dessa förenklingar kan skrivas

$$\frac{d^2}{dr^2}u(r) = F(r)u(r), \tag{2}$$

där $u(r)$ är deuteronens vågfunktion och $F(r)$ en funktion av r. Genom att summera Taylorutvecklingen av u i båda riktningar med steglängd h erhålls en numerisk lösningsformel

$$u_i = \frac{u_{i-1}(2 + \frac{5}{6}h^2F_{i-1}) - u_{i-2}(1 - \frac{1}{12}h^2F_{i-2})}{1 - \frac{1}{12}h^2F_i}, \tag{3}$$

för ekvation 2.

Vidare måste alla giltiga vågfunktioner, $u(r)$, uppfylla normaliseringsvillkoret

$$\langle u(r)|u(r)\rangle = \int_{-\infty}^{\infty} u(r)^* u(r)\, dr = 1. \tag{4}$$

Normaliseringsvillkoret ger således att $\int |u(r)|^2 dr < \infty$, vilket i sin tur kräver att $u(r) \to 0$ då $r \to 0$ och $u(r) \to 0$ då $r \to \infty$.

3 Metod

De numeriska värdena på konstanter erhölls från biblioteket `scipy.constants`, vilket importerades direkt till `Python`-koden [1]. För att i största mån undvika stora/små tiopotenser och därmed underlätta simuleringen valdes enheterna $1\,\mathrm{fm} = 1 \times 10^{-15}\,\mathrm{m}$ och $1\,\mathrm{MeV} \approx 1{,}602 \times 10^{-13}\,\mathrm{J}$ för dimensionerna längd och energi [2]. I detta fall valdes beräkna vågfunktionen på intervallet $r_{\min} \approx 0$ till $r_{\max} = 20\,\mathrm{fm}$.

Inledningsvis konstruerades en funktion för Malfliet-Tjon-potentialen enligt ekvation 1, där V_i och a_i var givna $\forall\, i$. Därefter skapades en loop för att iterera över deuteronens bindningsenergi E_d. Initialt sattes E_d till $E_d = \max\{V_{\mathrm{MT}}(r)\}/2$ och i varje iteration uppdaterades listan $F(r) = K(V(r) - E_d)$, där $K = 2m_n m_p/\hbar^2(m_n + m_p) \equiv 2\mu/\hbar^2$. För att konvertera K till enheter av fm och MeV användes omvandlingsfaktorn $\hbar c = 197{,}327\,\mathrm{MeV\,fm}$ [1]. Det gav $K = 2\mu/(\hbar c)^2$, där det krävs att μ uttrycks i $\mathrm{MeV/c^2}$.

Vågfunktionen delades upp i en utåt-integrerad, $u_<$, och en inåt-integrerad, $u_>$, del för att uppfylla randvillkoren i avsnitt 2. För att säkerställa vågfunktionens kontinuitet krävs att $u_<(r_*) = u_>(r_*)$ samt att derivatorna uppfyller $u'_<(r_*) = u'_>(r_*)$ i en matchningspunkt r_*. Ett villkor för detta härleddes från två Taylorutvecklingar kring r_* enligt

$$\begin{cases} u_>(r_* + h) = u_>(r_*) + hu'_>(r_*) + h^2 u''_>(r_*)/2 \\ u_<(r_* - h) = u_<(r_*) - hu'_<(r_*) + h^2 u''_<(r_*)/2. \end{cases}$$

Från Taylorutvecklingen löstes $u'_>(r_*)$ och $u'_<(r_*)$ ut och efter omskrivning med ekvation 2 erhölls

$$u'_>(r_*) - u'_<(r_*) = \frac{u_>(r_* + h) + u_<(r_* - h) - u(r_*)(2 - h^2 F(r_*))}{h} < \varepsilon,$$

där ε är en tolerans som sattes till $\varepsilon = 10^{-20}$. Vidare sattes $r_* = 1\,\mathrm{fm}$.

För att erhålla en giltig vågfunktion användes ekvation 4, där $\langle u(r)|u(r)\rangle$ beräknades med funktionen `integrate.trapzoid` från biblioteket `scipy` på intervallet $r_{\min} - r_{\max}$. Samma funktion användes även för att beräkna

$$\langle r^2\rangle = \langle u(r)|\, r^2\, |u(r)\rangle = \int_{r_{\min}}^{r_{\max}} u(r)^*\, r^2\, u(r)\, dr.$$

RMS-radien gavs sedan av $r_d = \sqrt{\langle r^2\rangle}/2$.

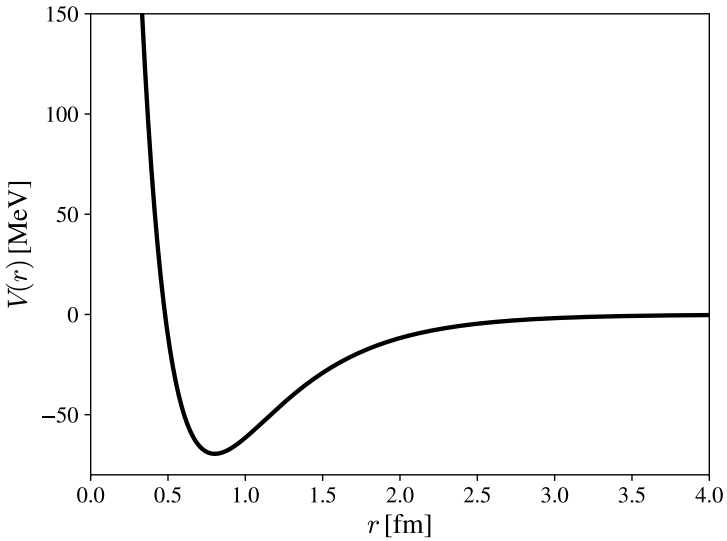
4 Resultat och diskussion

Malfliet-Tjon-potentialen (se ekvation 1) presenteras i figur 1. I figur 2 visas deuteronens normerade vågfunktion $u(r)$, vilken erhölls från simuleringen med Numerovs metod. För deuteronens RMS-radie och bindningsenergi erhölls $r_d = 1{,}935\,\mathrm{fm}$ respektive $E_d = -2{,}261\,\mathrm{MeV}$.

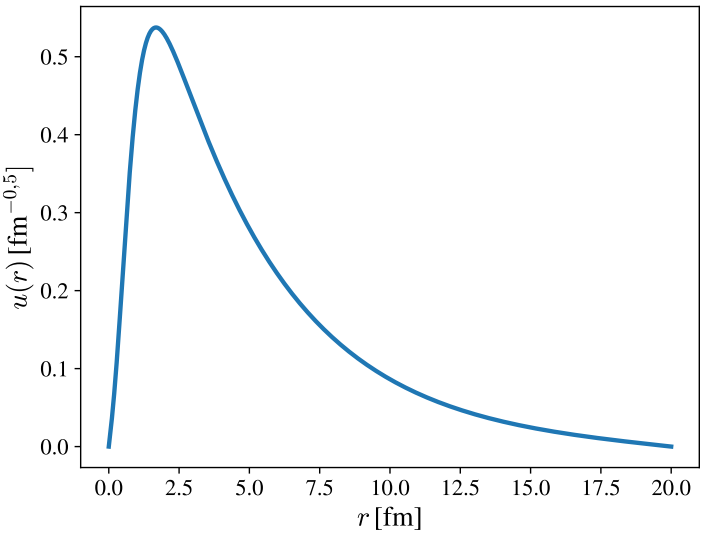
Den starka växelverkans korta räckvidd på några fm gör att deuteronens vågfunktion bör vara större på kortare avstånd. Att vågfunktionen dessutom går mot noll för små r är i också förväntat, eftersom neutronen och protonen har en viss utbredning. Den erhållna vågfunktionen i figur 2 är således rimlig.

Deuteronens RMS-radie och bindningsenergi har experimentellt uppmätts till $r_d = 1{,}97\,\mathrm{fm}$ och $E_d = -2{,}224\,\mathrm{MeV}$ [3]. De erhållna värdena från simuleringen stämmer relativt väl överens med dessa resultat. Vidare noteras att r_d ungefär sammanfaller med vågfunktionens maximum.

Om initialvärdet på E_d väljs negativt kommer lösningen konvergera till en giltig vågfunktion. Om E_d i stället sätts till ett större positivt värde ($E_d \gtrsim +5\,\mathrm{MeV}$) konvergerar inte metoden, vilket är rimligt då bindningsenergi *alltid* ska vara negativ.



Figur 1: Graf för Malfliet-Tjon-potentialen $V_{\mathrm{MT}}(r)$ som funktion av radien r.



Figur 2: I figuren visas den normerade vågfunktionen för deuteronen i en Malfliet-Tjon-potential.

Referenser

[1] The SciPy community. (2023), [Online]. Tillgänglig: https://docs.scipy.org/doc/scipy/reference/constants.html (hämtad 2023-05-15).

[2] C. Nordling och J. Österman, *Physics Handbook for Science and Engineering*, 9. utg. Lund, Sverige: Studentlitteratur, 2021, kap. C.U 2.4, s. 23, ISBN: 9780198520115.

[3] R. Pohl, F. Nez, L. M. P. Fernandes m. fl., "Laser spectroscopy of muonic deuterium", *Science*, årg. 353, nr 6300, s. 669–673, 2016. DOI: 10.1126/science.aaf2468. [Online]. Tillgänglig: https://www.science.org/doi/abs/10.1126/science.aaf2468.

Bilagor

A Källkod i Python

```
import numpy as np
import scipy.constants as K
from pyteomics import mass
import matplotlib.pyplot as plt
from scipy import integrate

# LaTeX font to plots
plt.rcParams['mathtext.fontset'] = 'cm'
plt.rcParams['font.family'] = 'STIXGeneral'

# --- Some constants ---
hbar = K.Planck / (2 * np.pi)
c = K.speed_of_light
e = K.elementary_charge
me = K.electron_mass
mp = K.proton_mass
mn = K.neutron_mass
fm = K.femto

# --- Theoretical values ---
Ed_exp = -2.224e6 * e
rd_ex = 1.97 * fm

# Malfliet-Tjon potential
def pot_MT(r):
    a1 = -586.04
    a2 = 1458.19
    a3 = -872.15
    mu1 = 1.55
    mu2 = 3.11
    mu3 = 6
    V1 = a1 * np.exp(-mu1 * r)
    V2 = a2 * np.exp(-mu2 * r)
    V3 = a3 * np.exp(-mu3 * r)
    return (V1 + V2 + V3) / r

# Conversion functions
def MeV_to_kg(m):
    M = 10**6 * m * e/c**2
    return M


def kg_to_MeV(m):
    M = m * c**2 / (10**6*e)
    return M


rmax = 20
N = 10000
h = rmax / N
mu = kg_to_MeV(mp * mn / (mp + mn)) # ţ in MeV
hc = hbar /(e * 1e6) * c /fm # conversion factor hc  197.327 MeV fm
```

```
K = 2 * mu / hc**2

r = np.linspace(1e-16, rmax, N+1)
u = np.zeros(N+1)
F = np.zeros(N+1)
Vr = np.zeros(N+1)

for i in range(0, N):
    Vr[i] = pot_MT(r[i])

# Parameters
Emin = np.min(Vr)
Emax = 0
E = 0.5 * (Emin + Emax)
max_iter = 1000
tol_kont = 1e-20


def find_nearest(array, value):
    array = np.asarray(array)
    idx = (np.abs(array - value)).argmin()
    return array[idx]


# Loop for iterating over energies E
for i in range(0, max_iter):
    for j in range(0, N):
        F[j] = K * (Vr[j] - E)


    # Choosing matching point where r  1 fm
    rmp_i = list(r).index(1)

    # Initialise out-integrated wave funciton
    u[0] = 0
    u[1] = h

    # Numerov out
    for j in range(2, rmp_i+1):
        u[j] = (u[j-1]*(2+5/6*h**2*F[j-1])-u[j-2]*(1-1/12*h**2*F[j-2]))/(1-1/12*h**2*F[j])
    u_out = u[rmp_i]

    #Initialise in-integrated wave funciton
    u[N] = 0
    u[N - 1] = h

    # Numerov inåt
    for j in range(N-2, rmp_i-1, -1):
        u[j] = (u[j+1]*(2+5/6*h**2*F[j+1])-u[j+2]*(1-1/12*h**2*F[j+2]))/(1-1/12*h**2*F[j])
    u_in = u[rmp_i]

    # Scale factors between wave funcitons
    scale = u_out / u_in

    # Matching the height of the inner and outer wave function
    u[rmp_i:N] = scale * u[rmp_i:N]

    # Calulating discontiuity at the matching point
    matchning = 1/h*(u[rmp_i-1]+u[rmp_i+1]-u[rmp_i]*(2+h**2*F[rmp_i]))
```

```
    # Update E in every loop
    if np.abs(matchning) < tol_kont:
        break
    if u[rmp_i] * matchning > 0:
        Emax = E
    else:
        Emin = E
    E = (Emin + Emax) / 2

u_int = integrate.trapezoid(u**2,r,h)
u = u/np.sqrt(u_int) # Normalized wave function

R = np.sqrt(integrate.trapezoid((u*r)**2,r,h))
rd = R / 2

rd = np.round(rd,4)
print(f'rd = {rd} fm')
E = np.round(E,4)
print(f'Ed = {E} MeV')


# Plotting resulting wave function and potential
font_size = 16
tick_size = 13
plt.figure(1)
plt.plot(r, u,linewidth = 2.5)
plt.xlabel('$r\,$[fm]',fontsize=font_size)
plt.ylabel('$u(r)\,$[fm$^{-0,\!5}]$',fontsize=font_size)
plt.xticks(fontsize = tick_size)
plt.yticks(fontsize = tick_size)

plt.figure(2)
plt.plot(r, Vr,linewidth = 2.5,color='black')
plt.xlim([0,4])
plt.ylim([-80,150])
plt.xlabel('$r\,$[fm]',fontsize=font_size)
plt.ylabel('$V(r)\,$[MeV]',fontsize=font_size)
plt.xticks(fontsize = tick_size)
plt.yticks(fontsize = tick_size)
plt.show()
```