Project 1

Alex Dombos, Samuel Lipschutz, Charles Loelius

January 22, 2014

1 ^{11}Be Structure

Based on a naive view of the shell model of the nucleus, the expected configuration of the 7 neutrons in the ground state of beryllium-11 would be $(1s_{\frac{1}{2}})^2(1p_{\frac{3}{2}})^4(1p_{\frac{1}{2}})^1$. However, for certain nuclei the neutron configuration may depend on the proton configuration [1]. In the case of beryllium-11, the $1s_{\frac{1}{2}}$ orbit is pushed down below the $1p_{\frac{1}{2}}$ orbit. Therefore, the correct configuration for the neutrons is $(1s_{\frac{1}{2}})^2(1p_{\frac{3}{2}})^4(2s_{\frac{1}{2}})^1$. We might naively argue that since the neutron is not tightly bound to the core, it must experience much less of the potential well, and so the centrifugal terms could come to dominate, thus leading the 2s term to come below some of the 1 p terms.

A (t,p) reaction was used to investigate the excited states of beryllium-11 [1][2]. Based on the ^{9}Be(t,p)^{11}Be reaction, the excited states from experiment are found in Table 1.

E_x (MeV ± keV)	Width (keV)	Configuration	J^π
0		$^{10}\mathrm{Be}(0^+)\otimes(2\mathrm{s}_{\frac{1}{2}})_\nu$	$\frac{1}{2}^+$
0.320 ± 2			$\frac{1}{2}^-$
1.784 ± 4	104 ± 21	$^{10}\mathrm{Be}(0^+)\otimes(1\mathrm{d}_{\frac{5}{2}})_\nu$	$\frac{5}{2}^+$
2.642 ± 9	228 ± 21		$\frac{3}{2}^-$
3.398 ± 6	104 ± 17	$^{9}\mathrm{Be}(0^+)\otimes(\mathrm{sd})^2_{0^+}$	$\frac{3}{2}^-$
3.888 ± 1		$^{10}\mathrm{Be}(2^+)\otimes(\mathrm{s}\frac{1}{2})$	$\frac{3}{2}^+$
3.955 ± 1		$^{9}\mathrm{Be}(0^+)\otimes(\mathrm{sd})^2_{2^+}$	$\frac{3}{2}^-$
5.255 ± 3		$^{9}\mathrm{Be}(0^+)\otimes(\mathrm{sd})^2_{2^+}$	$\frac{5}{2}^-$
5.849 ± 10	139 ± 17	$^{9}\mathrm{Be}(0^+)\otimes(\mathrm{sd})^2_{2^+}$	$\frac{1}{2}^-$

Table 1: ^{11}Be levels observed in the ^{9}Be(t,p)^{11}Be reaction

For example, the ground state occurs when the beryllium-10 core (in a $^+0$ state) is coupled to a $2s_{\frac{1}{2}}$ neutron (meaning the angular momentum quantum number, L, is

zero) and has a spin and parity of $J^\pi = \frac{1}{2}^+$; the first excited state occurs 0.320 MeV above the ground state and has a spin and parity of $J^\pi = \frac{1}{2}^-$; the second excited state occurs when the beryllium-10 core is coupled to a $1d_{\frac{5}{2}}$ neutron (meaning the angular momentum quantum number, L, is two), at 1.784 MeV, ans had a spin and parity of $J^\pi = \frac{5}{2}^+$. The second excited state is identified as a resonance that had a width of 104 keV.

2 Scattering Equations for $n - ^{10}Be$ with Arbitrary L

2.1 Radial Scattering Equations

We can write Schrodinger's equations as

$$H\psi = E\psi \tag{1}$$

Now using a basic separation of variables as applicable in spherical coordinates with azimuthal symmetry(which we have in this instance)

$$\psi = \Sigma_{l=0}^{\infty}(2L+1)i^L P_l(\cos\theta)\frac{1}{kr}\chi_L(r) \tag{2}$$

We can then just consider the r equation, leaving us with:

$$(-\frac{\hbar^2}{2\mu}(\frac{d^2}{dr^2} - \frac{L(L+1)}{r^2}) + V(r) - E)\chi_L(r) = 0 \tag{3}$$

Where we can then plug in for V:

$$\left(-\frac{\hbar^2}{2\mu}\left(\frac{d^2}{dr^2} - \frac{L(L+1)}{r^2}\right) + \frac{V_0}{1+e^{\frac{r-R_{ws}}{a_{ws}}}} - E\right)\chi_L(r) = 0 \tag{4}$$

We can represent this as:

$$\boxed{\chi''(r) = \frac{2\mu}{\hbar^2}\left(\frac{V_0}{1+e^{\frac{r-R_{ws}}{a_{ws}}}} - E\right)\chi + \frac{L(L+1)}{r^2}\chi} \tag{5}$$

2.2 Scattering Boundary Conditions

Now, if we plug in numbers for all of the constants, and make some assumptions for E and L, we are able to solve this numerically, if in addition we provide some sort of boundary conditions. We know that in order to prevent the terms from blowing up, this means that we need $\chi_L(0) = 0$, which is one such condition. We also need to have, for an elastic collision, the condition that at infinity we should expect the wavefunction to return to the form $e^{ikr+\delta}$. This can be written as at some chosen distance a, where we

would want $e^{ika+\delta}$, noting tangentially that k is determined by the (reduced) mass and energy of the neutron. This is explicitly:

$$k = \frac{p}{h} = \sqrt{\frac{2\mu E}{\hbar^2}} \tag{6}$$

From this we see that for any L value, we would simply match that

$$\chi(a) = H_L^- + S_L H_L^+ \approx \sin(ka - \frac{L\pi}{2} + \delta_L) \tag{7}$$

2.3 Numerical Solutions

We can write down the radial schoedinger equation for arbitrary central potential $V(|\vec{r}|) = V(R)$ as

$$u_L^{''}(R) = \left(\frac{L(L+1)}{R^2} + \frac{2\mu}{\hbar}(V(r) - E)\right) u_L(R) \tag{8}$$

where $u_L(R) = constant * \chi_L(R)$ and μ is the reduced mass of the system.

This is a second order linear equation, which can be solved numerially using a variety of methods. In our case we used fourth order Runge-Kutta (RK4) from the python Scipy package. Runge-Kutta is a means to sovle the general first order linear equation

$$y^{'} = f(t, y(t)) \tag{9}$$

Given initial condition $y(t_0) = y_0$ This is done by taking the expansing of $y(t+h)$ for small h and truncating at some order in h. The simplest method is to take the linear apparoximation (Euler method)

$$y(t+h) = y(t) + hy^{'}(t) \tag{10}$$

$$y(t+h) = y(t) + hf(t) \tag{11}$$

$$y_{n+1} = y_n + hf(t_n) \tag{12}$$

RK4 takes the terms up and including h^4 yeilding

$$y_{n+1} = y_n + \frac{1}{6}h(k_1 + 2k_2 + 2k_3 + k_4) \tag{13}$$

$$k_1 = f(t_n, y_n) \tag{14}$$

$$k_2 = f(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_1) \tag{15}$$

$$k_3 = f(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_2) \tag{16}$$

$$k_4 = f(t_n + h, y_n + hk_3) \tag{17}$$

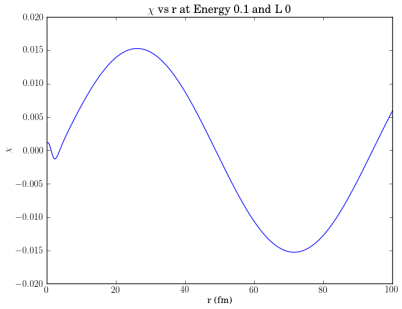
The k_1 term can be seen as just first order term (slope at the begining of the interval), while the k_2 term calculates the slope based on the midpoint found from the linear term. k_3 repeats this process again whereas k_4 takes the step based on the slope at the end of the interval using the previous term.
We can apply this to our scattering problem by breaking the second order equation into two coupled first order equations

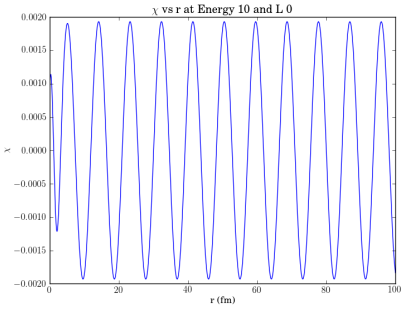
$$u_L^{''}(R) = \left(\frac{L(L+1)}{R^2} + \frac{2\mu}{\hbar}(V(r) - E)\right) u_L(R) \tag{18}$$

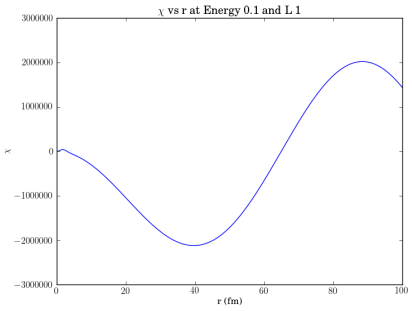
$$z_L^{'}(R) = \left(\frac{L(L+1)}{R^2} + \frac{2\mu}{\hbar}(V(r) - E)\right) u_L(R), z_L = u_L^{'} \tag{19}$$

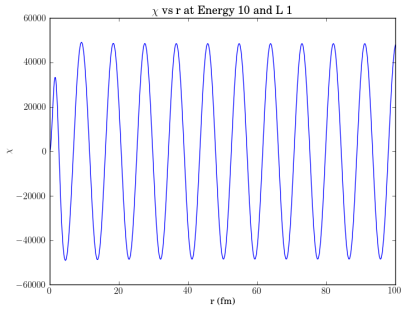
2.4 Plotting Radial Behaviour

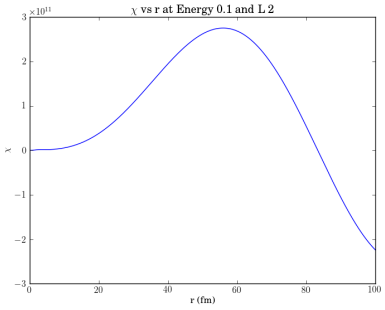
We have as follows the radial wave functions generated via the script at the end of this report:

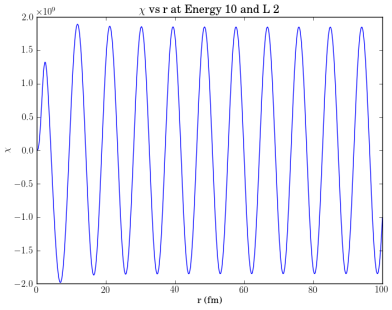




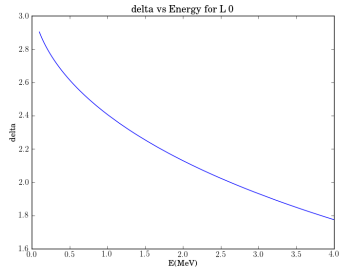


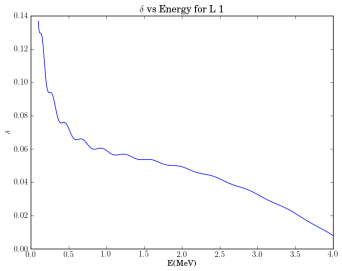


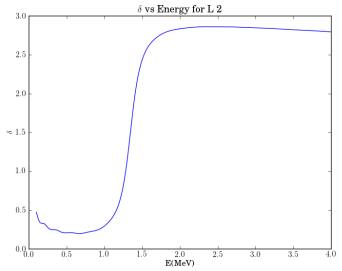




3 Delta Functions

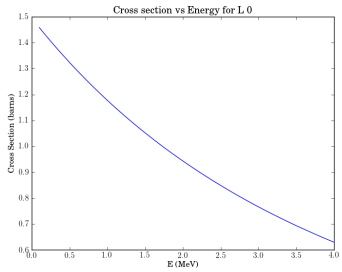


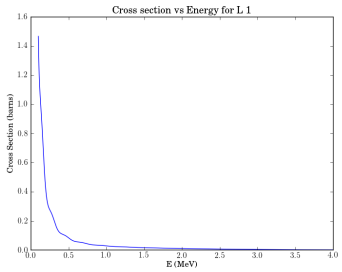


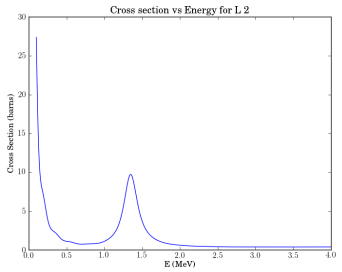


4 Resonances

While we can see resonances in the delta functions, most clearly in the L=2 case, we can see this even more clearly in the cross sections, once again calculated in the script attached, and which are plotted below.







A resonance can be identified as a peak in a plot of cross section against energy, or a sudden rise then leveling off in a plot of phase shift against energy. Based on our results shown in figures above a resonance can be identified for L=2, but not L=0 or L=1. The calculated peak was at 1.345 MeV, which compares favorably with the experimental result of 1.784 MeV. Similarly, the calculated width of the resonance at L=2 is 203 keV. This is in good agreement with the experimental results of the ^{9}Be(t,p)^{11}Be reaction, which had an observed resonance for L=2 with a width of 104 keV. These calculated values were found using the findpeakinfo method on the data generated from our numerical solutions.

5 References

1. I. Talmi, I. Unna, Phys. Rev. Lett. 4 (1960) 469

2. G.B. Liu, H.T. Fortune, Phys. Rev. C 42 (1990) 167

3. J.H. Kelley, E. Kwan, J.E. Purcell, C.G. Sheu, H.R. Weller, Nucl. Phys. A 88 (2012) 880

6 Appendix: Code for generating graphs

```
import scipy.integrate as integrate
import matplotlib.pyplot as plt
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
from scipy import special
from itertools import product
from scipy.optimize import fsolve, curve_fit
from scipy.interpolate import UnivariateSpline
from scipy import signal
plt.rc('text', usetex=True)
plt.rc('font', family='serif')
def findpeakinfo(xs,arr):
    "Script to find peak location and FWHM"
    a=signal.find_peaks_cwt(arr,np.linspace(1,2,10))
    r=a[1]
    print(xs[r])#Energy of peak, where we know first peak is at approximately 0.
    max=arr[r]
    values=[aa-max/2 for aa in arr]
    min=[0,-100]
    max=[len(values)-1,100]
    for i in range(50,r):
        if( abs(values[i])<abs(min[1])):
            min[0]=i
            min[1]=values[i]
    for i in range(r,len(values)):
        if( abs(values[i])<abs(max[1])):
            max[0]=i
            max[1]=values[i]
    print(min)
    print(max)
    return min,max
def makeanglecontinuous(angles):
    "Renormalize Angles to remove discontinuities by adding n*pi"
    count=0#how many times n pi we need to add/subtract
    outarray=[]
    for theta in angles:
        if(theta>=0):
            outarray.append(theta)
        else:
            outarray.append(theta+np.pi)
    return outarray

def diffeq(L,Energy):
    "This function returns a function that will be the differential equation to be solved based on E and L"
    def deriv_chi(f, r):
            chi, chiprime = f
            return [chiprime, (-2.9233295/(1.0+np.exp((r-2.58532)/.65))*chi-.047845*Energy*chi+L*(L+1)*chi/r**2) ]
    return deriv_chi
sin=np.sin
def HenkelPlus(rho,L):#Defining Hankel functions and their derivatives
    return np.exp(1j*(rho-L*np.pi/2))
def HenkelMinus(rho,L):
    return np.exp(-1j*(rho-L*np.pi/2))
def HenkelPlusPrime(rho,L,k):
    return k*1j*np.exp(1j*(rho-L*np.pi/2))
def HenkelMinusPrime(rho,L,k):
    return k*(-1j)*np.exp(-1j*(rho-L*np.pi/2))



class chi:
    "At some level a general second order differential equation solver, which anticipates the problem. It uses runge-kutta in
          SolveDifEq, after using SetDiffEq to make the equation to be solved. It keeps the result, as well as all input used in an
           object for easy calculation and plotting"
    diff=0#Differential equation to be solved, set via SetDiffEq
    init=[]#array of initial values in the form [y(0),y'(0)]
    rvalues=[]#array containing all r values to be sampled over
    Energy=0#Energy of system in Schrodinger Eq.
    L=0#Angular Momentum Quantum number in Schro. Eq.
```

```
    chivalues=[]#Array containing chi(x) values, each element in the array corresponds as such: chivalues[i]=chi(rvalues[i])
    chiprimevalues=[]#Similar to chivalues but for chi prime
    delta=0#Phase shift.
    Rmat=0#Logarithmic Derivative divided by a
    SMat=0#Represents admixture of Hankel plus function(which is to say "outgoing spherical wave")
    sindelta=0#sin of delta
    Mass=931.49272#Where this comes from taking .0478 from the notes, dividing by 2, multipling by hbar, and then dividing by (1 fm
          ^2*1MeV)
    crosssection=0#Cross Section in arb units.
    aindex=0#index of chivalues, chiprimevalues to be used to generate R matrix(i.e. chivalues[aindex]/chivaluesprime[aindex])
    def SetDiffEq(self):
        "Choose Differential Equation to be solved"
        self.diff=diffeq(self.L,self.Energy)
    def SolveDifeq(self):
        "Solve Differential Equation"
        z = integrate.odeint(self.diff, self.init, self.rvalues,full_output=0,mxstep=10000)
        self.chivalues, self.chiprimevalues=z.T
    def GetK(self):
        "Calculate k in fm"
        k=np.sqrt(.047845*self.Energy)#hbar^2k^2/2m=E
        return k
    def SetRMatrix(self):
        "Find Logarithmic Derivative"
        self.Rmat=self.chivalues[self.aindex]/self.chiprimevalues[self.aindex]/self.rvalues[self.aindex]
    def SetSMatrix(self):
        "Calculate S Matrix from RMatrix using Hankel Functions"
        k=self.GetK()
        a=self.rvalues[self.aindex]
        num=HenkelMinus(k*a,self.L)-a*self.Rmat*HenkelMinusPrime(k*a,self.L,k)
        denom=HenkelPlus(k*a,self.L)-a*self.Rmat*HenkelPlusPrime(k*a,self.L,k)
        self.SMat=num/denom
    def SetDelta(self):
        "Calculate Delta From Logarithm"
        self.delta=1/(2j)*np.log(self.SMat)
    def SetSinDelta(self):
        "Calculate Sin Delta, save in chi"
        self.sindelta=(sin(np.real(self.delta)))
    def SetCrossSection(self):
        "Calculate Cross Section in units of barns"
        self.crosssection=4*np.pi*(2*self.L+1)/(100*self.GetK()**2)*(self.sindelta)**2
    def __init__(self, rvalues,Energy,L,init,aindex):
        "Initialization, requires range, Energy, L, initial conditions and the index to be sampled at."
        self.rvalues=rvalues
        self.Energy=Energy
        self.L=L
        self.init=init
        self.aindex=aindex
def twodplot(x,y,title,xaxis,yaxis):
    fig=plt.figure()
    ax2=fig.add_subplot(111)
    ax2.plot(x,y)
    plt.title(title)
    plt.xlabel(xaxis)
    plt.ylabel(yaxis)


init=[.001,.001]
r=np.linspace(.0001,100,1000)
Energies=[.1,10]
Ls=[0,1,2]
radwaves=[]
ain=-10
for vars in product(Energies,Ls):
    radwav=chi(r,vars[0],vars[1],init,ain)
    radwav.SetDiffEq()
    radwav.SolveDifeq()
    radwav.SetRMatrix()
    radwav.SetSMatrix()
    radwav.SetDelta()
    radwav.SetSinDelta()
    twodplot(radwav.rvalues,radwav.chivalues,"$\chi$ vs r at Energy "+str(vars[0])+" and L "+str(vars[1]),"r (fm)", r"$\chi$")
    radwaves.append(radwav)
rs=np.linspace(100,10000,100)
EnergiesDelta=np.linspace(.1,4,1000)
for L in Ls:
    deltasen=[]
    sindeltasen=[]
    Smats=[]
    rmats=[]
    rmatsfromsmats=[]
    crosssections=[]
    i=0
    for En in EnergiesDelta:
```

```
        i=i+1
        radwav=chi(r,En,L,init,ain)
        radwav.SetDiffEq()
        radwav.SolveDifeq()
        radwav.SetRMatrix()
        radwav.SetSMatrix()
        radwav.SetDelta()
        radwav.SetSinDelta()
        radwav.SetCrossSection()
        deltasen.append(radwav.delta)
        sindeltasen.append(radwav.sindelta)
        Smats.append(radwav.SMat)
        rmats.append(np.abs(radwav.Rmat))
        crosssections.append(radwav.crosssection)
        rmatsfromsmats.append(1/radwav.rvalues[ain]*(HenkelMinus(radwav.GetK()*radwav.rvalues[ain],radwav.L)-radwav.SMat*HenkelPlus(
              radwav.GetK()*radwav.rvalues[-1],radwav.L))/((HenkelMinusPrime(radwav.GetK()*radwav.rvalues[-1],radwav.L,radwav.GetK
              ())-radwav.SMat*HenkelPlusPrime(radwav.GetK()*radwav.rvalues[-1],radwav.L,radwav.GetK())))
    twodplot(EnergiesDelta,makeanglecontinuous(deltasen),r" $\delta$ vs Energy for L "+str(L),"E(MeV)", r"$\delta$")
    twodplot(EnergiesDelta,crosssections," Cross section vs Energy for L "+str(L),"E (MeV)", " Cross Section (barns)")
    if(L==2):
        left,right=(findpeakinfo(EnergiesDelta,crosssections))
        print(EnergiesDelta[right[0]]-EnergiesDelta[left][0])#Prints fwhm of peak
plt.show()
```