PHY 480 Final Project: Dispersive Corrections to Optical Model Potential
Garrett King

Files

- dcomp_main.cpp – The main C++ file for the program.
- make_dcomp_main – The make file for dcomp_main.cpp. Running `make -f make_dcomp_main` will build the C++ code.
- run_dcomp.py – A python script built to run the code. The way this works is running `python run_dcomp.py <path/to/card/file>`, where the card file will have the same format as ExampleCard.in
- ExampleCard.in – File containing the parameters needed to construct the optical potential and control the numerics of the integrator and ode solver.
  - *Target/Projectile Z/A* – Charge/mass numbers of the target/projectile
  - *w1, w2, Ef+, Ef-, n* – The parameters needed for the Brown-Rho shape of a Koning-Delaroche [Nuc. Phys. A, 713(3-4), (2003).] type imaginary volume potential:

$$W_v(E) = -w_1 \frac{(E - E_f)^n}{(E - E_f)^n + w_2^n} \tag{1}$$

    Koning-Delaroche uses $n = 2$, but this parameter can be changed to change the behavior of the cross section in Energy, $\sigma(E)$. E_f is found by taking $\frac{1}{2}[E_f^+ - E_f^-]$. $E_f^\pm$ are the removal energy for the $A + 1$ and the target A nucleon systems. The values found in the ExampleCard.in were calculated using the NNDC chart of the nuclides values of the ground state energies for the A and $A \pm 1$ systems. w_1 and w_2 are the values for ^{40}Ca(p,p) given in Koning-Delaroche and $n = 2$ is the same as they used.
  - *HF real V, r, a* – The Hartree-Fock real volume, radius, and diffuseness parameters for the fixed real part of the potential. The ones in this file correspond to the Koning-Delaroche real volume parameters at the Fermi Energy of ^{40}Ca.
  - *Coulomb radius* – the radius that controls the behavior of the Coulomb potential. If this parameter is 0, no Coulomb term is included. Otherwise, inside the Coulomb radius the interaction corresponds to the projectile inside a charged sphere. Outside the radius, the interaction is the one between the projectile and a point charge with the charge of the target centered at the origin.
  - *Rmatch* – The matching radius to compare with asymptotics and find the S-matrix elements
  - *Lmax* – The maximum number of partial waves to be used in the calculation

  - *Npts* – Number of integration points for the RK4 stepper from `boost/numeric/odeint` used to solve scattering equation
  - *GSL integration abs_err, rel_err* – The absolute and relative error allowed for GSL integration to find dispersive corrections to real volume term

- Asym.* – header and C++ files for the Asym class. This class is called with the matching radius, a, the Sommerfeld parameter, η, the wave vector k, and L value of the partial wave and provides the L^{th} Hankel functions and their derivatives at this point

- OMP.* – header and C++ files for the OMP class. Takes in real and imaginary volume radius and diffuseness parameters and the target mass to find the optical potential. It has a function pot(double r) that finds the optical potential for these parameters at a point r.

- Coulomb.* – Takes in the charges of the target and projectile and the Coulomb radius to find the Coulomb potential. Function pot(double r) returns the Coulomb potential at a point r.

- plot.py – Code for plotting a cross section data file in python. To use it, run `python plot.py <path/to/cross/section/file>`. Output to `cross_sections.png`. Currently, the code has a line included to use latex for the axes. If this is causing problems, line 26 of the code can be commented out and the label text on lines 32 and 33 can be changed to say something else. Without line 26, those strings should still print out without being rendered for latex.

In addition to the headers that I wrote, the code depends on the GSL and Boost libraries for C++. Instructions for how to install Boost and properly build the code with these libraries are given in the next section.

The main code will output a file `cross_sections.out`, which contains total and reaction cross sections as a function of energy. The files `wf*` are the wave functions at different energies and partial waves. In the version included in Github and the tarball for the source code, the lines to print out the wave functions are commented out. To get the wave functions, uncomment lines 335 to 343, line 371, and line 391 and comment line 374. This will print a file for each partial wave at each energy used in the calculations. To view these partial waves, a code `view_partial_wave.py` is included. It is run like `plot.py`, but instead takes the partial wave file as an input argument.

Building Boost

In the directory containing my code, I included the file `boost_1_67_0.tar.gz`. Running `tar -xzvf boost_1_67_0.tar.gz` will untar the directory and have the files needed by the boost library. In `make_dcomp_main`, the path following `-I` in the definition of the `LIBS` variable on line 32 will need to be changed to point the top of the `boost_1_67_0` directory on your machine. When running the make file, the compiler will throw some warnings about

files in the boost library. The option that treats warnings as errors is not included so that the code could build even with those warnings being thrown.

Physics and function of the code

Using the input parameters for the fixed real ($\mathrm{Re}\,\Sigma(r, r', E_F)$) and imaginary volume ($\mathrm{Im}\,\Sigma(r, r', E')$) optical potential terms, a dispersive correction to the real part is evaluated:

$$\mathrm{Re}\,\Sigma(r, r'; E) = \mathrm{Re}\,\Sigma(r, r'; E_F) + \Delta\mathcal{V}(r, r'; E)$$

Where:

$$\Delta\mathcal{V} = \frac{1}{\pi}\Big(-\mathcal{P}\int_{E_f^+}^{\infty} dE'\,\mathrm{Im}\,\Sigma(r, r', E')\left[\frac{1}{E-E'} - \frac{1}{E_f - E'}\right] + \mathcal{P}\int_{-\infty}^{E_f^-} dE'\,\mathrm{Im}\,\Sigma(r, r', E')\left[\frac{1}{E-E'} - \frac{1}{E_f - E'}\right]\Big) \tag{2}$$

And $\mathcal{P}$ indicates the principal value integrals. Using the parameterization of $W_V(E)$ from Equation 1 and the fact that:

$$\mathrm{Im}\,\Sigma(r, r', E') = W_v(E') * f_{WS}(r, r', a)$$

, the dispersive correction is found in the code is found as follows:

$$\Delta\mathcal{V} = \frac{f_{WS}(r, r', a)}{\pi}\Big(-\mathcal{P}\int_{E_f^+}^{\infty} dE' W_v(E')\left[\frac{1}{E-E'} - \frac{1}{E_f - E'}\right] + \mathcal{P}\int_{-\infty}^{E_f^-} dE' W_v(E')\left[\frac{1}{E-E'} - \frac{1}{E_f - E'}\right]\Big) \tag{3}$$

With $f_{WS} = 1/[1+\exp(1+(r-r')/a)]$ being a Woods-Saxon form factor. The principal value integrals are evaluated using the QAWC integrator from the GSL Integration class. This integrator takes arguments $f(x)$ and c to evaluate $\int dx \frac{f(x)}{(x-c)}$. The integral was split into the four different singular integrals and the contributions were then summed up to get a volume term. The imaginary radius and diffuseness were used for the geometry terms. All together, the optical potential was:

$$U(r; E) = \mathrm{Re}\,\Sigma(r; E_F) + \mathrm{Im}\,\Sigma(r; E) + \Delta\mathcal{V}(r; E)$$

The odeint library from the Boost distribution was used to evaluate the Schrödinger Equation for different partial waves:

$$\left[\frac{-\hbar^2}{2\mu} + U(r;E) + \frac{\hbar^2 L(L+1)}{2\mu r^2} + V_C(r)\right]\chi_L(r) = E\chi_L(r)$$

With $V_C(r)$ being the Coulomb potential evaluated in the Coulomb class. A fourth order Runge-Kutta stepper was used with a specified number of integration points and an integration step dependent on the minimum and matching radii. The minimum radius should be set slightly off zero so the centrifugal term does not blow up the calculation. A loop was set up to go from $L = 0$ up to a maximum value of the angular momentum. The reason the Boost odeint library was chosen for this project over GSL was because Boost's solver could be adapted to handle complex numbers. The method involves defining a complex state $y(r)$ such that:

$$y(r) = \begin{pmatrix}\chi_L(r)\\ \phi_L(r)\end{pmatrix}$$

Where $\chi_L(r)$ and $\phi(r) = \chi'_L(r)$ are complex values of the partial wave and its derivative at r. To start, initial conditions must be provided. Physical requirements make $\chi_L(0) = 0$. Since the calculation does not start right a zero, a small value is given to the wave function at the minimum radius. Choosing $\phi_L(r_{min}) = p$, where p is some complex number, will provide the right answer for the wave function up to a normalization. Since the R and S matrix elements are the quantities of interest and come from an inverse logarithmic derivative at the matching radius, that normalization cancels out. The wave functions are evolved in space according to:

$$\begin{aligned}\frac{d\chi_L(r)}{dr} &= \phi_L(r)\\ \frac{d\phi_L(r)}{dr} &= \chi_L(r)\left[\frac{2\mu}{\hbar^2}(U(r;E) + V_C(r) - E) + \frac{L(L+1)}{r^2}\right]\end{aligned} \tag{4}$$

For wave L, the R-matrix element was found by taking the inverse logarithmic derivative at the matching radius, a:

$$R_L = \frac{1}{a}\frac{\chi_L(a)}{\chi'_L(a)}$$

And was used to find the S-matrix element:

$$S_L = \frac{H_L^-(\eta,ka) - aR_L H_L^{-\prime}(\eta,ka)}{H_L^+(\eta,ka) - aR_L H_L^{+\prime}(\eta,ka)}$$

Where $H_L^{\pm}$ are the Hankel functions. To make the calculations easier and avoid needing to multiply by $\hbar$ and c all throughout the computation, $\mu \to \frac{\mu}{\hbar^2}$ is used to in the code. In these units, $e^2 \to e^2/4\pi\varepsilon_0 = \hbar c/137 = 1.44$ MeV $\cdot$ fm can be used for the charge squared that appears in the Coulomb term and the Sommerfeld parameter, $\eta = e^2 Z_t Z_p k/(2E)$ with $k = \sqrt{2\mu E}$ in this convention.

The loop is also used to calculated the total (σ_{tot}) and reaction (σ_R) cross sections via:

$$\begin{aligned} \sigma_{el} &= \frac{10 \text{ mb}}{\text{fm}^2} \times \frac{\pi}{k^2} \sum_L (2L+1)|1 - S_L|^2 \\ \sigma_R &= \frac{10 \text{ mb}}{\text{fm}^2} \times \frac{\pi}{k^2} \sum_L (2L+1)(1 - |S_L|^2) \\ \sigma_{tot} &= \sigma_{el} + \sigma_R \end{aligned} \tag{5}$$

σ_{el} and σ_{tot} only have meaning for a neutral projectile, so the plotting code gives options to look at the total cross section or the reaction cross section, depending on which type of projectile you are looking at.

Results

The results of this code for ^{40}Ca(p,p) and ^{40}Ca(n,n) are shown below in Figure 1. In the figure, the reaction cross section is plotted using `plot.py`. For a charged projectile, the reaction cross sections should be plotted. For a neutral projectile, one should look at the total cross section. The sorts of results we are comparing in with are seen in Figure 2, taken from M. Atkinson et al. PRC 98, 044627 (2018) [`https://journals.aps.org/prc/abstract/10.1103/PhysRevC.98.044627`]. There, the authors show different fits of their optical model against data for these reactions. While they use an optical potential with more complicated effects than the one in this code, one should still expect to see similar shapes coming from the toy model predictions. For both reactions, the general features of the more sophisticated model are captured. For neutron scattering, the drastic dip at low energies, followed by a smooth bump in the cross section is seen. For proton scattering case, the cross sections rise steadily to the top, and then begin to descend back down. Changing parameters will alter how these features look in the prediction, but the general trends are still present. Results are shown for different parameter sets to show how the observable would be impacted by changes to the model.

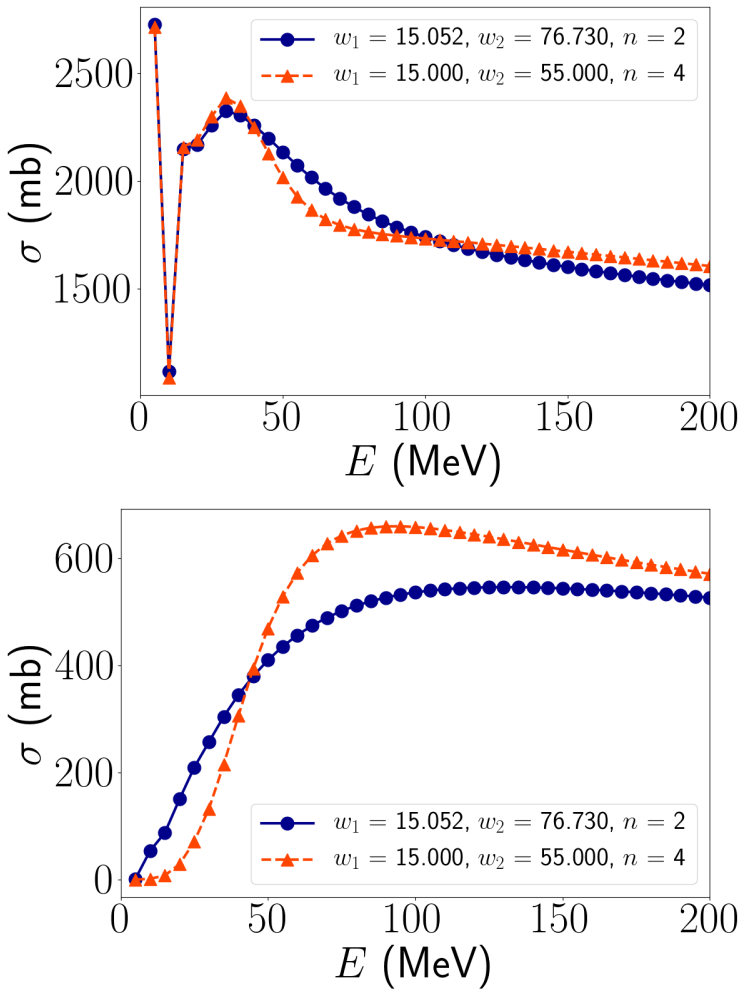

Figure 1: ^{40}Ca(n,n) (TOP) and ^{40}Ca(p,p) (BOTTOM) scattering from the toy model used in this project.

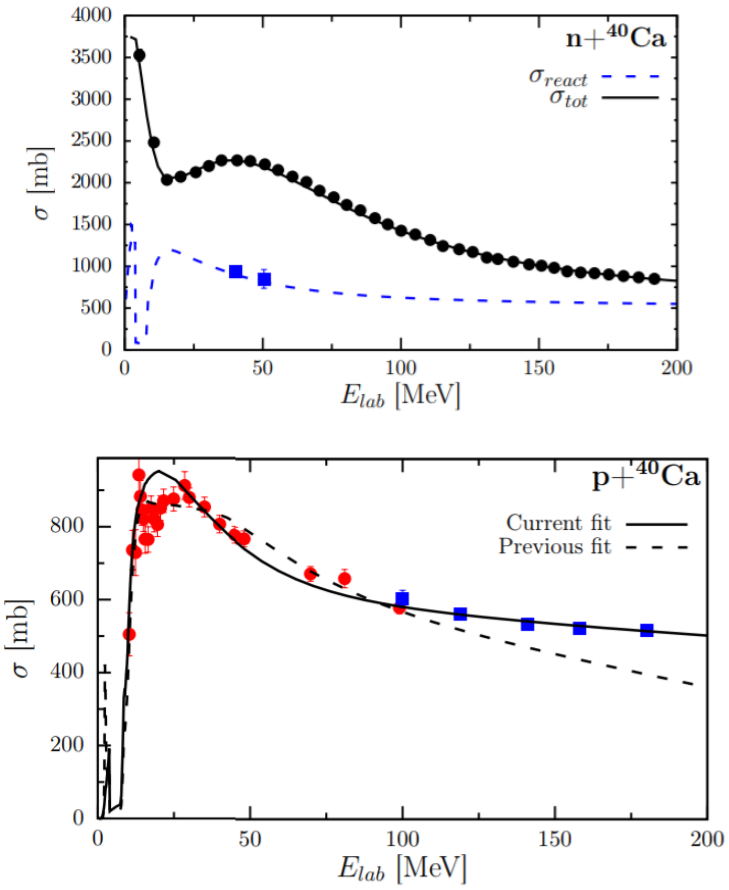

Figure 2: ^{40}Ca(n,n) (TOP) and ^{40}Ca(p,p) (BOTTOM) scattering from M. Atkinson et al. PRC 98, 044627 (2018).