Project 1, deadline November 17

Nuclear Forces PHY989

National Superconducting Cyclotron Laboratory and Department of Physics and Astronomy, Michigan State University, East Lansing, USA

Fall semester 2017

Nucleon-nucleon scattering

The aim of this project is to solve the Lippman-Schwinger equation for two interacting nucleons and relate the obtained phase shifts with those extracted from the experimental cross sections.

We are going to solve the Schroedinger equation (SE) for the neutron-proton system in momentum space for positive energies E in order to obtain the phase shifts. We can rewrite the SE in momentum space as

$$\frac{k^2}{m}\psi_l(k) + \frac{2}{\pi}\int_0^\infty dqq^2 V_l(k,q)\psi_l(q) = E\psi_l(k). \tag{1}$$

The derivation is covered by the lecture notes on scattering theory. This derivation is discussed during the regular lectures.

Here we have used units $\hbar = c = 1$. This means that k has dimension energy. k is the relative momentum between the two particles. A partial wave expansion has been used in order to reduce the problem to an integral over the magnitude of momentum only. The subscript l refers therefore to a partial wave with a given orbital momentum l. Below we will let the interaction to couple different values of l. This leads to what is called a coupled-channel problem. For details, see again the lecture notes on scattering theory.

To obtain the potential in momentum space we used the Fourier-Bessel transform (Hankel transform)

$$V_l(k,k') = \int j_l(kr)V(r)j_l(k'r)r^2dr, \tag{2}$$

where j_l is the spherical Bessel function. We will just study the case $l = 0$, which means that $j_0(kr) = sin(kr)/kr$.

For scattering states, $E > 0$, the corresponding equation to solve is the so-called Lippman-Schwinger equation. This is an integral equation where we have to deal with the amplitude $R(k,k')$ (reaction matrix) defined through the integral equation

$$R_l(k,k') = V_l(k,k') + \frac{2}{\pi}\hat{P}\int_0^\infty dqq^2 V_l(k,q)\frac{1}{E-q^2/m}R_l(q,k'), \tag{3}$$

where the total kinetic energy of the two incoming particles in the center-of-mass system is

$$E = \frac{k_0^2}{m}. \tag{4}$$

The symbol $\hat{P}$ indicates that Cauchy's principal-value prescription is used in order to avoid the singularity arising from the zero of the denominator. We will discuss below how to solve this problem. Eq. (3) represents then the problem you will have to solve numerically.

The matrix $R_l(k,k')$ relates to the the phase shifts through its diagonal elements as

$$R_l(k_0,k_0) = -\frac{tan\delta_l}{mk_0}. \tag{5}$$

From now on we will drop the subscript l in all equations.

In order to solve the Lippman-Schwinger equation in momentum space, we need first to write a function which sets up the mesh points. We need to do that since we are going to approximate an integral through

$$\int_a^b f(x)dx \approx \sum_{i=1}^N w_i f(x_i),$$

where we have fixed N lattice points through the corresponding weights w_i and points x_i.

Project 1a): Setting up the integration domain, mesh points and weights. Start writing your main program by setting up the mesh points and the corresponding weights. Fix first the number of mesh points N. Use either the Fortran or C++ programs which allow you to set up integration weights and points using Gaussian quadrature, that is get the weights ω_i and the points k_i. Before you go on you need to keep in mind that if you use Legendre polynomials (the function GaussLegendreQuadrature), the Legendre polynomials are defined in the interval $x \in [-1,1]$. Your integral is for the interval $r \in [0,\infty)$. You will need to map the weights from the function GaussLegendreQuadrature (C++ version) to your interval. To do this, call first GaussLegendreQuadrature(a,b,x[],w[],N) (similar links for the Fortran version), with $a=-1$, $b=1$. It returns the mesh points x_i and weights w_i. You map these points over to the limits in your integral. You can then use the following mapping

$$k_i = const \times tan\left\{\frac{\pi}{4}(1+x_i)\right\},$$

and

$$\omega_i = const\frac{\pi}{4}\frac{w_i}{cos^2\left(\frac{\pi}{4}(1+x_i)\right)}.$$

If you opt for units fm^{-1} for k, set $const = 1$. If you opt to work with MeV, set $const \sim 200$ ($\hbar c = 197$ MeVfm). You must decide which units to use.

Project 1b): Adding a potential model. The next step is to write a function which calculates the potential in momentum space. The potential we will use here is a parametrized potential between a proton and neutron for the partial wave ${}^1S_0$, i.e., spin $S = 0$ and orbital momentum $l = 0$, a singlet S-state. This state does not have a bound state for the deuteron (only the triplet S-state has). The parametrized version of this potential fits the experimental phase-shifts. It is given by

$$V(r) = V_a\frac{e^{-ax}}{x} + V_b\frac{e^{-bx}}{x} + V_c\frac{e^{-cx}}{x} \tag{6}$$

with $x = \mu r$, $\mu = 0.7$ fm^{-1} (the inverse of the pion mass), $V_a = -10.463$ MeV and $a = 1$, $V_b = -1650.6$ MeV and $b = 4$ and $V_c = 6484.3$ MeV and $c = 7$. Find the potential in momentum space using Eq. (2) with $j_0(kr) = sin(kr)/kr$. The transform of a potential on the form $V(r) = V_0\exp(-\mu r)/r$ is

$$V(k',k) = \frac{V_0}{4k'k}ln\left(\frac{(k'+k)^2+\mu^2}{(k'-k)^2+\mu^2}\right). \tag{7}$$

Write a function which calculates the expressions for the potential in momentum space.

Project 1c): Handling the principal value problem. The principal value in Eq. (3) is rather tricky to evaluate numerically, mainly since computers have limited precision. We will here use a subtraction trick often used when dealing with singular integrals in numerical calculations. We introduce first the calculus relation

$$\int_{-\infty}^{\infty}\frac{dk}{k-k_0} = 0. \tag{8}$$

It means that the curve $1/(k-k_0)$ has equal and opposite areas on both sides of the singular point k_0. If we break the integral into one over positive k and one over negative k, a change of variable $k \to -k$ allows us to rewrite the last equation as

$$\int_0^{\infty}\frac{dk}{k^2-k_0^2} = 0. \tag{9}$$

We can use this to express a principal values integral as

$$\mathcal{P}\int_0^{\infty}\frac{f(k)dk}{k^2-k_0^2} = \int_0^{\infty}\frac{(f(k)-f(k_0))dk}{k^2-k_0^2}, \tag{10}$$

where the right-hand side is no longer singular at $k = k_0$, it is proportional to the derivative df/dk, and can be evaluated numerically as any other integral.

We can then use the trick in Eq. (10) to rewrite Eq. (3) as

$$R(k,k') = V(k,k') + \frac{2}{\pi}\int_0^\infty dq \frac{q^2 V(k,q)R(q,k') - k_0^2 V(k,k_0)R(k_0,k')}{(k_0^2 - q^2)/m}. \quad (11)$$

This is the equation you are going to solve numerically in order to calculate the phase shifts of Eq. (5).We are interested in obtaining $R(k_0, k_0)$.

How do we proceed in order to solve Eq. (11)?

The first step consists in using the mesh points k_j and the weights ω_j. We can rewrite Eq. (11) as

$$R(k,k') = V(k,k') + \frac{2}{\pi}\sum_{j=1}^{N} \frac{\omega_j k_j^2 V(k,k_j)R(k_j,k')}{(k_0^2 - k_j^2)/m} - \frac{2}{\pi}k_0^2 V(k,k_0)R(k_0,k')\sum_{n=1}^{N}\frac{\omega_n}{(k_0^2 - k_n^2)/m}. \quad (12)$$

This equation contains now the unknowns $R(k_i, k_j)$ (with dimension $N \times N$) and $R(k_0, k_0)$. We can turn Eq. (12) into an equation with dimension $(N+1)\times(N+1)$ with a mesh which contains the original mesh points k_j for $j = 1, N$ and the point which corresponds to the energy k_0. Consider the latter as the 'observable' point. The mesh points become then k_j for $j = 1, n$ and $k_{N+1} = k_0$.

With these new mesh points we define the matrix

$$A_{i,j} = \delta_{i,j} - V(k_i, k_j)u_j, \quad (13)$$

where δ is the Kronecker δ and

$$u_j = \frac{2}{\pi}\frac{\omega_j k_j^2}{(k_0^2 - k_j^2)/m} \qquad j = 1, N \quad (14)$$

and

$$u_{N+1} = -\frac{2}{\pi}\sum_{j=1}^{N}\frac{k_0^2\omega_j}{(k_0^2 - k_j^2)/m}. \quad (15)$$

The first task is then to set up the matrix A for a given k_0. This is an $(N + 1) \times (N + 1)$ matrix. It can be convenient to have an outer loop which runs over the chosen observable values for the energy k_0^2/m. *Note that all mesh points k_j for $j = 1, N$ must be different from k_0. Note also that $V(k_i, k_j)$ is an $(N + 1) \times (N + 1)$ matrix.* Write a small function which sets up A.

With the matrix A we can rewrite Eq. (12) as a matrix problem of dimension $(N + 1) \times (N + 1)$. All matrices R, A and V have this dimension and we get

$$A_{i,l}R_{l,j} = V_{i,j}, \quad (16)$$

or just

$$AR = V. \tag{17}$$

Since you already have defined A and V (these are stored as $(N+1) \times (N+1)$ matrices) Eq. (17) involves only the unknown R. We obtain it by matrix inversion, i.e.,

$$R = A^{-1}V. \tag{18}$$

Thus, to obtain R, you will need to set up the matrices A and V and invert the matrix A. To do that you can use the examples on matrix inversion (this link brings you to the c++ version). With the inverse A^{-1}, performing a matrix multiplication with V results in R.

With R you can then evaluate the phase shifts by noting that

$$R(k_{N+1}, k_{N+1}) = R(k_0, k_0), \tag{19}$$

and you are done.

You can choose to read k_0 from file or screen, or set up a loop over chosen values of k_0 and for each k_0 solve Eq. (18).

When you have $R(k, k')$ for the given potential, evaluate now the phase-shifts using

$$R(k_0, k_0) = -\frac{tan\delta}{mk_0}.$$

Compare the phase shifts for the potential of Eq. (6) with the experimental phase shifts that can be found in the article of the Nijmegen group in Physical Review C **48**, 792 (1993). Alternatively look up their website

Project 1d): Variable Phase Approach. Here we explore a simple alternative to the momentum-space matrix inversion for the calculation of scattering phase shifts called the Variable Phase Approach (VPA). The VPA (in its simplest formulation) is not as flexible as the matrix inversion method in that it is limited to local potentials (i.e., $\langle \mathbf{r}'|V|\mathbf{r}\rangle = \delta^3(\mathbf{r}-\mathbf{r}')V(r)$) without tensor forces. What the VPA lacks in generality, it makes up for in simplicity and the ability to better control the numerical accuracy. The VPA also gives a clean way to infer certain properties of the interaction from the phase shifts (e.g., the existence of a repulsive short-range component, or the presence of bound states) as illustrated below. For simplicity, here we only consider s-waves. Good references here are

1. Taylor, *Scattering Theory*, pages 197-201, and
2. Calogero, *The Variable Phase Approach to Potential Scattering*, (Academic Press, New York, 1967).

The questions which follow can be solved through a mix of standard analytical work and programming.

Define the truncated potential $V_\rho(r)$ by

$$V_\rho(r) = V(r)\theta(\rho - r)\ .$$

That is, it is the usual potential for $r \leq \rho$, but identically zero beyond that. Then we define $\delta(k, \rho)$ as the phase shift for V_ρ at momentum k. The phase shift we want is $\delta(k) = \lim_{\rho \to \infty} \delta(k, \rho)$. The basis of the variable phase method is a differential equation for $\delta(k, r)$ at fixed k (again, this is the s-wave equation):

$$\frac{d\delta(k,r)}{dr} = -\frac{1}{k} 2MV(r) \sin^2[kr + \delta(k,r)],$$

which is a nonlinear first-order differential equation with initial condition $\delta(k, 0) = 0$. Note that M here is actually the reduced mass of the NN system, and also ask yourself if there are factors of $\hbar$ and/or c that have been set to 1. Think about how you would implement this in your favorite programming language. As an example, the Mathematica notebook SquareWellScattering.nb implements the VPA for a square well. Show that it reproduces the analytically known phase shifts for the square well result.

Changing to a different potential is trivial (see the illustration at the end of the notebook with a combined short-range repulsive square well and a mid-range attractive square well). Implement the toy NN potential that you are using in your momentum space matrix inversion code as a check on the former.

Show from the VPA differential equation that a fully attractive potential gives a positive phase shift and a fully negative potential gives a negative phase shift. This is the cleanest way to see why the s-wave phaseshifts (which change from positive to negative values at $E_{lab} \approx 270$ MeV in the $^1\mathrm{S}_0$ partial wave) imply a strong short-range repulsion for local NN potentials.

The VPA automatically builds in Levinson's theorem ($\delta(0) = n\pi$) about the number of bound states n and the phase shift at zero energy. How?

Hint. Hint: what is the condition imposed on the phase shift at large energy for Levinson's theorem? Consider integrating $d\delta(k, r)/dr$ in r from zero to infinity. Use $\sin^2 x \leq 1$ to put a bound on $\delta(k)$.

Things to try numerically with the supplied Mathematica or Python notebooks:

1. Try out Levinson's theorem in practice (e.g., for a square well where the number of bound states versus depth is easily found in parallel). Also, the toy NN potential as given does not support a bound state since it describes scattering in the $^1S_0$ channel, though the large negative scattering length indicates that there is "almost" a bound state. By gently adjusting the strength of the longest ranged component of the force, estimate the critical strength for when a bound state first appears.

   (a) Explore the effective range expansion by extracting the a and r_0 parameters for 2 different functional forms of $V(r)$ (e.g., the toy NN potential and the square well potential). Then, try to tune the square well potential so it gives the same ERE parameters as the other one. This illustrates that there is no "unique" potential insofar as low-energy data is concernced.

Introduction to numerical projects

Here follows a brief recipe and recommendation on how to write a report for each project.

- Give a short description of the nature of the problem and the eventual numerical methods you have used.
- Describe the algorithm you have used and/or developed. Here you may find it convenient to use pseudocoding. In many cases you can describe the algorithm in the program itself.
- Include the source code of your program. Comment your program properly.
- If possible, try to find analytic solutions, or known limits in order to test your program when developing the code.
- Include your results either in figure form or in a table. Remember to label your results. All tables and figures should have relevant captions and labels on the axes.
- Try to evaluate the reliabilty and numerical stability/precision of your results. If possible, include a qualitative and/or quantitative discussion of the numerical stability, eventual loss of precision etc.
- Try to give an interpretation of you results in your answers to the problems.
- Critique: if possible include your comments and reflections about the exercise, whether you felt you learnt something, ideas for improvements and other thoughts you've made when solving the exercise. We wish to keep this course at the interactive level and your comments can help us improve it.
- Try to establish a practice where you log your work at the computerlab. You may find such a logbook very handy at later stages in your work, especially when you don't properly remember what a previous test version of your program did. Here you could also record the time spent on solving the exercise, various algorithms you may have tested or other topics which you feel worthy of mentioning.

Format for electronic delivery of report and programs

The preferred format for the report is a PDF file. You can also use DOC or postscript formats or as an ipython notebook file. As programming language we prefer that you choose between C/C++, Fortran2008 or Python. The following prescription should be followed when preparing the report:

- Use your github repository to upload your report. Indicate where the report is by creating for example a **Report** folder. Please send us as soon as possible your github username.

- Place your programs in a folder called for example **Programs** or **src**, in order to indicate where your programs are. You can use a README file to tell us how your github folders are organized.
- In your git repository, please include a folder which contains selected results. These can be in the form of output from your code for a selected set of runs and input parameters.
- In this and all later projects, you should include tests (for example unit tests) of your code(s).
- Comments from us on your projects, with score and detailed feedback will be emailed to you.

Finally, we encourage you to work two and two together. Optimal working groups consist of 2-3 students. You can then hand in a common report.