16. 780.20 Session 16

a. Bound States in Momentum Space

The ordinary time-independent Schrödinger equation in coordinate space for a local potential is an ordinary differential equation:

\[- \frac{\nabla^2}{2 \mu} \psi_n(r) + V(r) \psi_n(r) = E_n \psi_n(r),\]  \hspace{1cm} (16.1)

where \(\mu\) is the reduced mass (which is \(M/2\) if we are considering two interacting particles of mass \(M\) each). For scattering states, where \(E_n > 0\), any choice of \(E_n\) will give an acceptable solution (assuming \(V(r) \to 0\) as \(r \to \infty\)). For bound states, only discrete values of \(E_n\) yield normalizable wave functions, so we have an eigenvalue problem. In the more general (and less familiar case), the potential is non-local and we have an integro-differential equation to solve:

\[- \frac{\nabla^2}{2 \mu} \psi_n(r) + \int d^3r' V(r, r') \psi_n(r') = E_n \psi_n(r).\]  \hspace{1cm} (16.2)

In momentum space, the equation for the momentum space wave function \(\psi_n(k)\) is always an integral equation. Consider the abstract Schrödinger equation,

\[\hat{H} |\psi_n\rangle = \left(\frac{\hat{P}^2}{2 \mu} + \hat{V}\right) |\psi_n\rangle = E_n |\psi_n\rangle.\]  \hspace{1cm} (16.3)

Now hit this on the left with \(\langle k|\) and insert

\[1 = \int d^3k' |k'| \langle k'| \]  \hspace{1cm} (16.4)

to obtain

\[\frac{k^2}{2 \mu} \langle k| \psi_n \rangle + \int d^3k' \langle k| V(k, k') |k'\rangle \langle k'| \psi_n \rangle = E_n \langle k| \psi_n \rangle\]  \hspace{1cm} (16.5)

or, in an alternative notation for the same thing,

\[\frac{k^2}{2 \mu} \psi_n(k) + \int d^3k' V(k, k') \psi_n(k') = E_n \psi_n(k).\]  \hspace{1cm} (16.6)

If we expand in a partial wave basis (check your favorite quantum book!), then the resulting one-dimensional equation in the \(l^{th}\) partial wave takes the form

\[\frac{k^2}{2 \mu} \psi_n(k) + \frac{2}{\pi} \int_0^\infty V(k, k') \psi_n(k') k'^2 dk' = E_n \psi_n(k),\]  \hspace{1cm} (16.7)

where \(k \equiv |k|\) and we omit \(l\) labels.
The potential in partial waves is the “Bessel transform” of the full potential (why not the Fourier transform?):

\[ V(k, k') = \int_0^\infty r \, dr \int_0^\infty r' \, dr' \, j_l(kr) V(r', r) j_l(k' r) , \]

(16.8)

which reduces for a local potential to

\[ V(k, k') = \int_0^\infty r^2 \, dr \, j_l(kr) V(r) j_l(k' r) . \]

(16.9)

Recall that the first two spherical Bessel functions are

\[ j_0(z) = \frac{\sin z}{z} , \quad j_1(z) = \frac{\sin z}{z^2} - \frac{\cos z}{z} , \]

(16.10)

so for \( l = 0 \), the potential is simply

\[ V(k, k')_{l=0} = \frac{1}{kk'} \int_0^\infty dr \sin(kr) V(r) \sin(k'r) . \]

(16.11)

b. Numerical Solution

So how do we solve for the \( E_n \)'s and corresponding \( \psi_n(k) \)'s in Eq. (16.7)? As we’ve done before, we discretize it (that is, break up the continuous range in \( k \) into mesh points) and turn it into a matrix eigenvalue problem. Thus, if we have an integration rule (such as Gaussian quadrature) that performs an integral from 0 to \( \infty \) as a sum over \( N \) points \( \{k_i\} \) with weights \( \{w_i\} \), then the integral over the potential becomes

\[ \int_0^\infty k^2 \, dk' V(k, k') \psi_n(k') \approx \int_{j=0}^{N-1} w_j k_j^2 V(k, k_j) \psi_n(k_j) . \]

(16.12)

Thus the Schrödinger equation becomes

\[ \frac{k_i^2}{2\mu} \psi_n(k_i) + \frac{2}{\pi} \sum_{j=0}^{N-1} w_j k_j^2 V(k_i, k_j) \psi_n(k_j) = E_n \psi_n(k_i) , \quad i = 0, \ldots, N - 1 . \]

(16.13)

This is just the matrix problem

\[ \sum_j H_{ij} [\psi_n]_j = E_n [\psi_n]_j , \]

(16.14)

with

\[ H_{ij} = \frac{k_i^2}{2\mu} \delta_{ij} + \frac{2}{\pi} V(k_i, k_j) k_j^2 w_j , \quad i, j = 0, \ldots, N - 1 . \]

(16.15)

We can turn this over to a packaged matrix eigenvalue routine and get the eigenvalues and eigenvectors directly.

Note, however, that the matrix is not symmetric, so we can’t use the simple GSL routines. Instead we’ll use a general eigenvalue solver from the LAPACK subroutine library. There are versions of LAPACK for C and C++, but the most robust version is written in Fortran. So we’ll use this problem as an excuse to see how to call Fortran routines from C++. 
c. Delta-Shell Potential

The potential we’ll use in this session is the “delta-shell” potential, which in the coordinate representation is

$$ V(r) = \frac{\lambda}{2\mu} \delta(r - b) $$

(16.16)

where $\mu$ is the reduced mass of the particles interacting via $V$ (or just think of $\mu$ as the mass of a particle in the external potential $V$). Note that this is not a delta function at the origin; the potential is zero unless the particles are separated precisely by a distance $r = b$. So if we have a force that effectively acts over a very short range of distances, this would be a reasonable caricature. Besides the mass, the parameters are the range $b$ and the strength $\lambda$. From Eq. (16.16) you should be able to directly determine the units of $\lambda$.

The s-wave ($l = 0$) Schrödinger equation has (at most) one bound-state (that is, $E < 0$) solution. If we define $\kappa$ by writing the bound-state energy as

$$ E = -\frac{\kappa^2}{2\mu} $$

(16.17)

the value of $\kappa$ is determined by the solution to the transcendental equation

$$ e^{-2\kappa b} - 1 = \frac{2\kappa}{\lambda} \quad (l = 0) . $$

(16.18)

For general $l$, the bound-state $\kappa$ is the solution to

$$ 1 - \frac{\lambda}{i\kappa} (i\kappa b)^2 j_l(i\kappa b) \left[ n_l(i\kappa b) - i j_l(i\kappa b) \right] . $$

(16.19)

Can you derive either of these results? Is there always one bound state?

The delta-shell potential is trivial to convert to momentum space:

$$ V(k', k) = \int_0^\infty r^2 dr \, j_l(k'r) \frac{\lambda}{2\mu} \delta(r - b) j_l(kr) = \frac{\lambda b^2}{2\mu} j_l(k'b) j_l(kb) , $$

(16.20)

where $l$ is the angular momentum state we are considering. Note that this is not a very well-behaved function in momentum space! That means you may have to be clever in doing a numerical integral. The wave function of the $l = 0$ bound state in coordinate space is

$$ \psi_0(r) = \int_0^\infty k^2 dk \, \psi_0(k) j_0(kr) \propto \begin{cases} e^{-\kappa r} - e^{\kappa r}, & \text{for } r < b , \\ e^{-\kappa r}, & \text{for } r > b . \end{cases} $$

(16.21)

d. Calling Fortran from C++

We’ll use the example of calling a LAPACK fortran library machine from C or C++.

1. Add an underscore to the lowercase name of the Fortran routine, e.g., DGEEV becomes dgeev_.

2. Include a prototype declaration of the Fortran subroutine. Put `extern "C" { }` around the prototype. For example:

```c
extern "C" {
    void dgeev_(char* JOBVL, char* JOBVR, int* dimension,
                double Hmat_passed[], int* dimension, double Eigval_real[],
                double Eigval_imag[], double Eigvec_left[][1], int* LDVL,
                double Eigvec_right_passed[], int* LDVR, double WORK[],
                int* LWORK, int* INFO);
}
```

3. The variable type used in C++ should match the Fortran type, as illustrated in this chart (which uses the variable definitions in the Session 16 code `deltashell_boundstates.cpp`):

<table>
<thead>
<tr>
<th>Fortran</th>
<th>C++</th>
</tr>
</thead>
<tbody>
<tr>
<td>defined</td>
<td>passed</td>
</tr>
<tr>
<td>characterized*1 JOBVL</td>
<td>JOBVL</td>
</tr>
<tr>
<td>INTEGER N</td>
<td>N</td>
</tr>
<tr>
<td>REAL<em>8 WR(</em>)</td>
<td>WR</td>
</tr>
<tr>
<td>double* Eigval_real</td>
<td>double Eigval_real[]</td>
</tr>
<tr>
<td>= new double [dimension]</td>
<td>Eigval_real</td>
</tr>
</tbody>
</table>

(Note: There is an alternative way of making these calls in C++, that calls by value rather than reference. See me if you want an example.)

4. Fortran arrays start at 1 rather than 0. This is not a problem when passing arrays from C++ to Fortran. Simply fill the C++ array as usual (starting from 0), pass the pointer to Fortran, and it will be interpreted as starting from 1.

5. The Fortran array element A(3,5) is A[4][2] in C++ (subtract one for zero base indexing and reverse the order of subscripts). You must fill your C++ arrays accordingly.

6. Compile the C++ parts as usual and then link using

   `-lm -lblas -llapack -lg2c`

   (For libraries other than LAPACK, link with the appropriate library names; you’ll always need `-lg2c`, however.)

---

e. Dynamically Allocating Space for Arrays

Suppose we want to allocate f[i] with space for `maxsize` elements. Then:

```c
double* f = new double [maxsize]
```

after which we can refer to `f[0], f[1], ..., f[maxsize-1]`. To deallocate `f` and free the memory:

```c
delete [] f
```

(note that no number appears between the []'s). See the `deltashell_boundstates.cpp` code for an example of how to allocate and deallocate two-dimensional arrays.

f. References
