# 16
# Quantum Eigenvalues via ODE Matching

*In this chapter we examine how a differential-equation solver may be combined with a search algorithm to solve the eigenvalue problem. This problem requires us to solve the bound-state eigenvalue problem for the 1D, time dependent Schrödinger equation. Even though this equation is an ODE, which we know how to solve quite well by now, the extra requirement that we need to solve for* bound states *makes this an* eigenvalue problem*. Specifically, the bound-state requirement imposes a boundary conditions on the form of the solution, which, in turn, means that a solution will exist only for certain energies, the* eigenenergies *or eigenvalues.*

*If this all sounds a bit much for you now, rest assured that you do not need to understand all the physics behind these statements. What we want, is for you to get experience with the technique of conducting a numerical search for the eigenvalue, in conjunction with solving an ODE numerically. This is how one solve the numerical, ODE eigenvalue problem. In Section 29.1.1, we discuss how to solve the equivalent, but more advanced, momentum-space eigenvalue problem as a matrix problem. In Chap. 25 we study the related problem of the motion of a quantum wave packet confined to a potential well. Further discussions of the numerical bound-state problem are found in [28, 29].*

**Problem:** We want to determine whether the rules of quantum mechanics are applicable inside of a nucleus. More specifically, you are told that nuclei contain neutrons and protons ("nucleons") with mass $mc^2 \simeq 940\,\text{MeV}$, and that a nucleus has a size of about 2 fm.[1] Your explicit **problem** is to see if these experimental facts are compatible, first, with quantum mechanics and, second, with the observation that there is a typical spacing of several million electron volts (MeV) between the ground and excited states in nuclei.

[1] A fm, or fermi, equals $10^{-13}$ cm = $10^{-15}$ m, and $\hbar c \simeq 197.32$ MeV fm.

*Computationyal Physics. Problem Solving with Computers (2nd edn).*
Rubin H. Landau, Manuel José Páez, Cristian C. Bordeianu

ISBN: 978-3-527-40626-5

## 16.1 Theory: The Quantum Eigenvalue Problem

Quantum mechanics describes phenomena that occur at atomic or subatomic scales (an elementary particle is subatomic). It is a statistical theory in which the probability that a particle is located at point $x$ is $\mathcal{P} = |\psi(x)|^2\,dx$, where $\psi(x)$ is called the *wave function*. If a particle of energy $E$ moving in one dimension experiences a potential $V(x)$, its wave function is determined by the ODE known as the time-independent Schrödinger equation:[2]

$$\frac{-\hbar^2}{2m}\frac{d^2\psi(x)}{dx^2} + V(x)\psi(x) = E\psi(x). \tag{16.1}$$

In addition, when our problem tells us that the particle is "bound," we are being told that it is confined to some finite region of space. The mathematical expression of confinement is that the probability of finding the particle over all of space is 1:

$$\int_{-\infty}^{\infty} dx\,|\psi(x)|^2 = 1 \tag{16.2}$$

Yet the only way to have a $\psi(x)$ with finite integral is to have it decay exponentially as $x \to \pm\infty$:

$$\psi(x) \to \begin{cases} e^{-x} & \text{for } x \to +\infty \\ e^{+x} & \text{for } x \to -\infty \end{cases} \tag{16.3}$$

In summary, although it is straightforward to solve the ODE (16.1) with the techniques we have learned so far, we must also require that the solution $\psi(x)$ simultaneously satisfies the boundary conditions (16.3). This extra condition turns the ODE problem into an *eigenvalue problem* that has solutions (*eigenvalues*) only for a certain values of the energy $E$. The ground state energy corresponds to the smallest (most negative) eigenvalue. The ground state wave function (eigenfunction), which we must determine in order to find its energy, must be nodeless and even (symmetric) about $x = 0$. The excited states have higher (less negative) energies, and wave functions that may be odd (antisymmetric).

### 16.1.1 Model: Nucleon in a Box

The numerical methods we describe are capable of handling the most realistic potential shapes. Yet to make a connection with the standard textbook case,

[2] The time-dependent equation requires the solution of a partial differential equation, as discussed in Chap. 25.

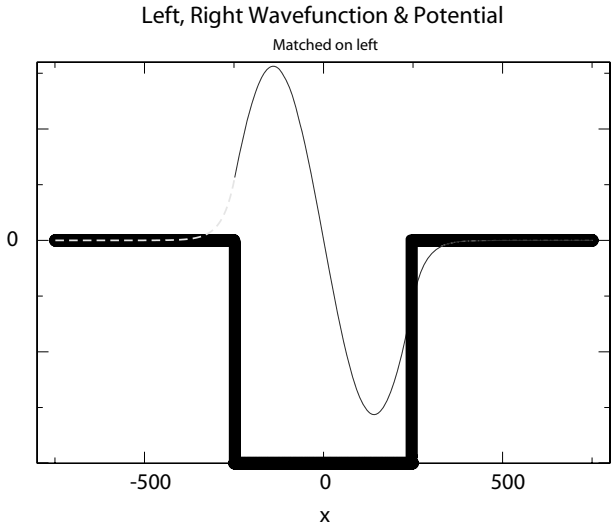


**Fig. 16.1** Computed wave function and square well potential. The wave function computed by integration in from the right is matched to the one computed by integration in from the left (dashed) at a point near the left edge of the well. Note how the wave function decays rapidly outside of the well.

and to permit some analytic checking, we will use a simple model in which the potential $V(x)$ in (16.1) is a finite square well (Fig. 16.1):

$$V(x) = \begin{cases} -V_0 = -83\,\text{MeV} & \text{for } |x| \le a = 2\,\text{fm} \\ 0 & \text{for } |x| > a = 2\,\text{fm} \end{cases} \tag{16.4}$$

A depth of 83 MeV and radius of 2 fm are typical for nuclei, and therefore this problem will be solved with energies in millions of electron volts and lengths in fermis. With this potential the Schrödinger equation (16.1) becomes

$$\frac{d^2\psi(x)}{dx^2} + \frac{2m}{\hbar^2}(E + V_0)\psi(x) = 0 \quad \text{for } |x| \le a \tag{16.5}$$

$$\frac{d^2\psi(x)}{dx^2} + \frac{2m}{\hbar^2}E\psi(x) = 0 \quad \text{for } |x| > a \tag{16.6}$$

To evaluate the ratio of constants here, we insert $c^2$, the speed-of-light squared, into the both numerator and the denominator [30, Appendix A.1]:

$$\frac{2m}{\hbar^2} = \frac{2mc^2}{(\hbar c)^2} \simeq \frac{2 \times 940\,\text{MeV}}{(197.32\,\text{MeV fm})^2} = 0.0483\,\text{MeV}^{-1}\text{fm}^{-2} \tag{16.7}$$

### 16.1.2 Algorithm: Eigenvalues via ODE Solver + Search

The solution of the eigenvalue problem combines the numerical solution of the ordinary differential equation (16.5) with a trial-and-error search for a wave function that satisfies the boundary conditions (16.3). This is done in several steps:

1. Start on the very far *left*, that is, at $x = -X_{\max} \simeq -\infty$, where $X_{\max} >> a$, the width of the potential. We assume that the wave function there satisfies the LH boundary condition:

$$\psi_L(x = -X_{\max}) = e^{+x(=-X_{\max})} = e^{-X_{\max}}$$

2. Use your favorite ODE solver to step $\psi_L(x)$ in toward the origin (to the right) from $x = -X_{\max}$, until you reach the *matching radius* $x_{\text{match}}$. The exact value of this matching radius is not important, and our final solution should be independent of it. In Fig. 16.1, we show a sample solution with $x_{\text{match}} = -a$, that is, we match at the left edge of the potential well. In Fig. 16.2 we see some guesses that do not match.

3. Start on the very far *right*, that is, at $x = +X_{\max} \simeq +\infty$, with a wave function that satisfies the RH boundary condition:

$$\psi_R(x = +X_{\max}) = e^{-x(=X_{\max})} = e^{-X_{\max}}.$$

4. Use your favorite ODE solver to step $\psi_R(x)$ in toward the origin (to the left) from $x = +X_{\max}$, until you reach the *matching radius* $x_{\text{match}}$. This means that we have stepped through the potential well (Fig. 16.1).

5. In order for probability and current to be continuous, $\psi(x)$ and $\psi'(x)$ must be continuous at $x = x_{\text{match}}$. Requiring the ratio $\psi'(x)/\psi(x)$, called the *logarithmic derivative*, to be continuous encapsulates both continuity conditions into a single condition, and is independent of $\psi$'s normalization.

6. Even though we do not know ahead of time which energies $E$ are eigenvalues, we still need a value for the energy in order to use our ODE solver. Such being the case, we start off the solution with a guess for the energy. A good guess for the ground state energy would be a value somewhat up from the bottom of the well.

7. Because it is unlikely that any guess will be correct, the left- and right-wave functions will not quite match at $x = x_{\text{match}}$ (Fig. 16.2). This is okay because we can use the amount of mismatch to improve the next

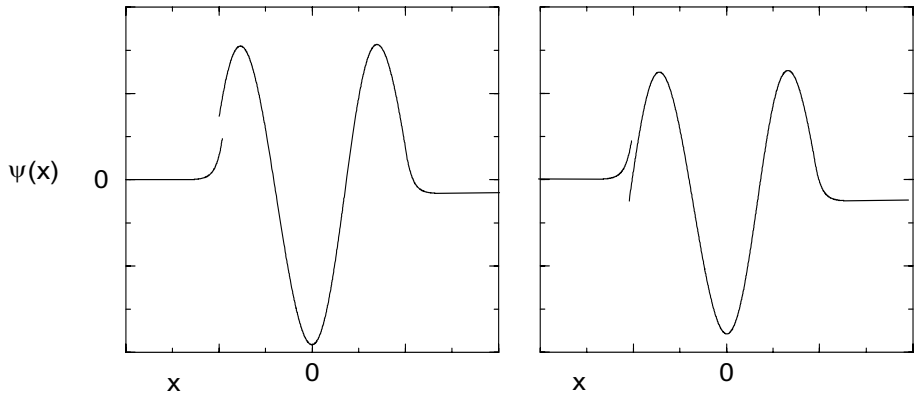


**Fig. 16.2** *Left:* A first guess at a wave function with an energy $E$ that is 0.5% too low. We see that the left wave function does not vary rapidly enough to match the right one at $x = 500$. *Right:* A second guess at a wave function with an energy $E$ that is 0.5% too high. We see that, now, the left wave function varies too rapidly.

guess. We measure how well the right and left wave functions match by calculating the difference

$$\Delta(E) = \left. \frac{\psi_L'(x)/\psi_L(x) - \psi_R'(x)/\psi_R(x)}{\psi_L'(x)/\psi_L(x) + \psi_R'(x)/\psi_R(x)} \right|_{x=x_{\text{match}}} \tag{16.8}$$

where the denominator is used to avoid overly large or small numbers. Then we try a different energy, note how much $\Delta(E)$ has changed, and then make an intelligent guess at an improved energy. The search continues until the left- and right-wave logarithmic derivatives match within some level of tolerance.

**Listing 16.1:** QuantumEigen.java solves the 1D, time-independent Scrödinger equation for bound state energies using the rk4 algorithm.

```
// QuantumEigen.java: solves Schroed eq via rk4 + Bisection Algor

import java.io.*;

public class QuantumEigen  {
  static double eps = 1E-6 ;              // Class variables, precision
  static int n_steps = 501;                          // Number int steps

  public static void main(String[] argv)
                        throws IOException, FileNotFoundException {

    double E = -17., h = 0.04;    // Initial E in MeV, step size in fm
    double Emax, Emin, Diff;
    int count, count_max = 100;

    Emax = 1.1*E;      Emin = E/1.1;
```

```
                                                    // Iteration loop
  for ( count=0;  count <= count_max;  count++ ) {
    E = (Emax + Emin)/2. ;                                  // Bisection
    Diff = diff(E, h);
    System.out.println("E = " + E + ", L-R Log deriv(E) = " +
        Diff);
                                                // Bisection algorithm
    if (diff(Emax, h)*Diff > 0) Emax = E;
    else Emin = E;
    if ( Math.abs(Diff)  <  eps ) break;
  }
  plot(E, h);
  System.out.println("Final eigenvalue E = " + E);
  System.out.println("iterations, max = " +count+", " + count_max);
  System.out.println("WF in QuantumL/R.dat, V in QuantumV.dat ");
}                                                           // End main

public static double diff(double E, double h)       // L-R log deriv
                    throws IOException, FileNotFoundException {
  double left, right, x;
  int ix, nL, nR, i_match;
  double y[] = new double[2];
  i_match = n_steps/3;                                // Matching radius
  nL = i_match + 1;
  y[0] = 1.E-15;                                    // Initial wf on left
  y[1] = y[0]*Math.sqrt(-E*0.4829);
                                                            // Left wf
  for ( ix = 0;  ix  <  nL + 1;  ix++ )  {
    x = h * (ix  -n_steps/2);
    rk4(x, y, h, 2, E);
  }
  left = y[1]/y[0];                                   // Log derivative
  y[0] = 1.E-15;              // - slope for even;  reverse for odd
  y[1] = -y[0]*Math.sqrt(-E*0.4829);
                                                      // Initialize R wf
  for ( ix = n_steps ;   ix > nL + 1;  ix--)  {
    x = h * (ix + 1 -n_steps/2);
    rk4(x, y, -h, 2, E);
  }
  right = y[1]/y[0];                                   // Log derivative
  return( (left-right)/(left + right) );
}

   // Repeat integrations for plot, can't integrate out decaying wf
public static void plot(double E, double h)
                      throws IOException, FileNotFoundException {
  PrintWriter L =
       new PrintWriter(new FileOutputStream("QuantumL.dat"), true);
  PrintWriter R =
       new PrintWriter(new FileOutputStream("QuantumR.dat"), true);
  PrintWriter Vx =
       new PrintWriter(new FileOutputStream("QuantumV.dat"), true);
  double left, right, normL, x = 0.;
  int ix, nL, nR, i_match;
  double y[] = new double[2];  double yL[][] = new double [2][505];
  int n_steps = 1501;                  // Total no integration steps
  i_match = 500;                                      // Matching point
```

```
    nL = i_match + 1;
    y[0] = 1.E-40;                              // Initial wf on the left
    y[1] = -Math.sqrt(-E*0.4829) *y[0];
    for ( ix = 0;  ix <= nL;  ix++ ) {
      yL[0][ix] = y[0];
      yL[1][ix] = y[1];
      x = h * (ix  -n_steps/2);
      rk4(x, y, h, 2, E);
    }                                           // Integrate to the left
    y[0] = -1.E-15;                 // - slope: even;  reverse for odd
    y[1] = -Math.sqrt(-E*0.4829)*y[0];
    for ( ix = n_steps -1;  ix >= nL + 1;  ix--)  {  // Integrate in
      x = h * (ix + 1 -n_steps/2);
      R.println(x + "  " + y[0] + "  " + y[1]);          // File print
      Vx.println(x + "  " + 1.E34*V(x));                   // Scaled V
      rk4(x, y, -h, 2, E);
    }
    x = x  - h;
    R.println(x + "  " + y[0] + "  " + y[1]);          // File print
    normL = y[0]/yL[0][nL];           // Renormalize L wf & derivative
    for ( ix = 0;  ix <= nL;  ix++ )  {
      x = h * (ix-n_steps/2 + 1);
      y[0] = yL[0][ix]*normL;
      y[1] = yL[1][ix]*normL;
      L.println(x + "  " + y[0] + "  " + y[1]);          // File print
      Vx.println(x + "  " + 1.E34*V(x));                   // Print V
    }
      return;
  }

   public static void f(double x, double y[], double F[], double E)
     { F[0] = y[1];     F[1] = -(0.4829)*(E-V(x))*y[0];  }

  public static double V(double x)
    { if (Math.abs(x)  <  10.)  return ( -16.);   else return(0.) ; }

                                                          // rk4 algorithm
  public static void rk4(double t, double y[], double h,
            int Neqs, double E)  {
    int i;
    double F[] = new double[Neqs];
    double ydumb[] = new double[Neqs];
    double k1[] = new double[Neqs]; double k2[] = new double[Neqs];
    double k3[] = new double[Neqs]; double k4[] = new double[Neqs];

    f(t, y, F,E);
    for (i=0; i<Neqs; i++)
        { k1[i] = h*F[i];
         ydumb[i] = y[i] + k1[i]/2;}

    f(t + h/2, ydumb, F,E);
    for (i=0; i<Neqs; i++)
        { k2[i] = h*F[i];
         ydumb[i] = y[i] + k2[i]/2;}

    f(t + h/2, ydumb, F,E);
        for (i=0; i<Neqs; i++)
```

```
        { k3[i]=  h*F[i];
          ydumb[i] = y[i] + k3[i];}

    f(t + h, ydumb, F,E);
    for (i=0; i<Neqs; i++)
    { k4[i] = h*F[i];
      y[i] = y[i] + (k1[i] + 2*(k2[i]+k3[i]) + k4[i])/6;}
  }
}
```

### 16.1.3 Implementation: Eigenvalues via ODE Solver + Bisection

1. Combine your bisection algorithm search program with your `rk4` ODE solver program. Start with a step size $h = 0.04$.
2. Write a subroutine which calculates the matching function $\Delta(E)$ as a function of energy and matching radius. This subroutine will be called by the bisection algorithm program to search for the energy at which $f(E, x = 2)$ vanishes.
3. As a first guess, take $E \simeq 65$ MeV.
4. Search until $\Delta(E)$ changes in the fourth decimal place, and modify the **f** function in `rk4` as appropriate for the Schrödinger equation (16.5). We do this in the code `QuantumEigen.java` shown in Listing 16.1.
5. Print out the value of the energy for each iteration. This will give you a feel as to how well the procedure converges, as well as a measure of the precision obtained. Try different values for the tolerance until you are confident that you are obtaining three good decimal places in the energy.
6. Build in a limit to the number of energy iterations you permit, and print out when the iteration scheme fails.
7. As we have done in Fig. 16.1, plot the wave function and potential on the same graph (you will have to scale the potential to get them both to fit).
8. Deduce, by counting the number of nodes in the wave function, whether the solution found is a ground state (no nodes) or an excited state (with nodes), and whether the solution is even or off (the ground state must be even).
9. Include in your version of Fig. 16.1, a horizontal line within the potential indicating the energy of the ground state relative to the potential's depth.

10. Increase the value of the initial energy guess and search for excited states. Make sure to examine the wave function for each state found to ensure that it is continuous, and to count the number of nodes. The number of nodes should increase as the levels get higher.

11. Add each new state found as another horizontal bar within the potential.

12. Verify that you have solved the **problem**, that is, that the spacing between levels is on the order of MeV for a nucleon bound in a several-fm well

### 16.1.4 Explorations

1. Check to see how well your search procedure works by using arbitrary values for the starting energy. For example, because no bound-state energies can lie below the bottom of the well, try $E = -V_0$ as well as some arbitrary fractions of $V_0$. In every case examine the resulting wave function and check that it is both symmetric and continuous.

2. Increase the depth of your potential progressively until you find more than one bound state. Look at the wave function in each case and correlate the number of nodes in the wave function and the position of the bound state in the well.

3. Explore how a bound-state energy changes as you change the depth $V_0$ of the well. In particular, as you keep decreasing the depth, watch the eigenenergy move closer to $E = 0$, and see if you can find the potential depth at which the bound state has $E \simeq 0$.

4. For a fixed well depth $V_0$, explore how the energy of a bound state changes as the well radius $a$ is varied.

5. Conduct some explorations in which you discover different values of $(V_0, a)$ that give the same ground-state energies. The existence of several different combinations means that knowledge of a ground-state energy is not enough to determine a unique depth of the well.

6. Modify the procedures to solve for the eigenvalue and eigenfunction for odd wave functions.

7. Solve for the wave function of a linear potential:

$$V(x) = -V_0 \begin{cases} |x| & \text{for } |x| < a \\ 0 & \text{for } |x| > a \end{cases}$$

There is less potential here than for a square well, so you may expect lower binding energies and a less-confined wave function. (For this potential, there are no analytic results with which to compare.)

8. Compare the results obtained, and the time the computer took to get them, with the Numerov method and with `rk4`.