CP Activity 5

Energy Levels of the Finite Square Well

Hand-out: 9 October 2023 **Hand-in:** 16 October 2023

5.1 Basic Task

In this activity, we will find the energy and plot the corresponding wave function of a particle in the ground state of a finite square well.

5.2 Introduction

We will consider a particle, an "electron", bound in a 1-dimensional finite square well potential of depth $-V_0 = -40.0$ eV and width $2a = 0.1$ nm (i.e. domain bounded in $-a \leq x \leq a$).

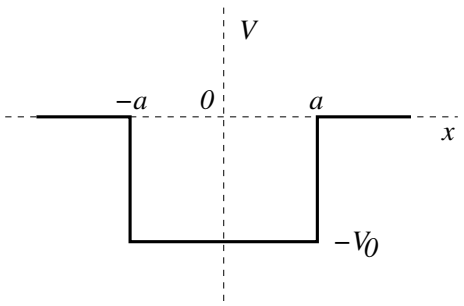

Figure 5.1: The potential for the particle in a box

We want to solve the time independent Schrödinger equation

$$-\frac{\hbar^2}{2m}\frac{d^2}{dx^2}\psi_0(x) + V(x)\,\psi_0(x) = E_0\,\psi_0(x)$$

for the ground state energy $E_0 = -\varepsilon$ and the wave function $\psi_0(x)$.

The reduced mass of the system is $mc^2 = 0.511$ MeV.
A useful value is $\hbar c = 197.3$ eV nm.
(Note: it is usually more useful to do calculations in units scaled to the problem, here eV and nm. Note that $\hbar^2/2m = (\hbar c)^2/2mc^2$, so you don't need to do conversions to SI units).

The equation to be solved[1] for the case of the 1-d particle in the box is

$$l\tan(la) - \kappa = 0 \tag{5.1}$$

with $l = \sqrt{2m(V_0 - \varepsilon)/\hbar^2}$ and $\kappa = \sqrt{2m\varepsilon/\hbar^2}$.

Note that V_0 and ε have been defined so that they are *positive*; this avoids some confusion with sign, but might not avoid others.

1. Eq. 5.1 can be solved numerically in the form $F(\varepsilon) = 0$, in order to find the bound state energy $E_0 = -\varepsilon_0$. Obviously, $0 < \varepsilon < V_0$.

   A brute force grid-search with $\delta E = 0.5$ eV is good enough as a first go (i.e. search until you find values of the function on either side of zero). Hint: search for $F(\varepsilon)F(\varepsilon + \delta E) < 0$ (why?). You can then use the bisection method discussed later to refine the estimate.

[1] You will need to work through the discussion in the textbook to see exactly how this equation arises.

2. Are there any more bound states in this potential? You can determine this without computation, but you may need to work through the discussion in the textbook. Hint: don't forget to consider the odd/even parity of the solutions and how this affects Eq. 5.1?

3. How strong must the potential be in order to have the same binding energy as the hydrogen atom, $E_0 = -\varepsilon = -13.6$ eV?

   You need to solve the same equation, but now with V_0 as the variable, rather than ε.

4. Have a look at the more advanced methods below and try one, e.g. secant or false position. How do your answers change?

   This can be a little tricky, since we can't expect the function *ever* to have an exact zero in floating point arithmetic. A simple implementation is given here. Note that we can use multiple assignments for swapping values around (see Section 5.5).

```
# search for f(x)=0 within some tolerance
tolerance=1.0e-10  # fractional uncertainty in root
dx = 0.1 # step size : tune this
x0 = ... # starting point
f0 = f(x0)
x1 = x0+dx
f1 = f(x1)
while 1:
    dx = -f1*dx/(f1-f0)
    x2 = x1+dx
    f2 = f(x2)
    if abs((x2-x1)/x2) < tolerance : break
    x1,f0,f1 = x2,f1,f2  # save for next iteration
```

   Note how the exit from the loop occurs.
   Of course, this is not programmed very defensively: it is possible that the loop might run forever if the tolerance is too tight for instance. We also don't want `x2` to become zero.

5. Plot out the ground state wave function. You will have to think about the matching condition at the well edge in order to get the scale right.

5.3 Root Searching: Some Methods

A basic problem in numerical analysis is to find the roots of an equation $f(x) = 0$. So, we can both capitalise on the vast amount of work done on this (from the time of Newton!) and learn something new and important, beyond the current application to quantum mechanics.

The following examples are very simple methods of root-finding:

- *Grid search.* This is a brute force search method. Search on a grid of spacing Δx for two values of x, say x_i and x_{i+1} which bracket the root. A finer grid can then be set up and the procedure repeated.

- *Binary search (bisection method).* Find two values of x, say x_1 and x_2 which bracket the root, and then reduce Δx by a factor of two each step while still bracketing the root.

Note that both of these methods (in fact *all* methods) find an approximate value for the root, so we will need some criterion to know if we've gone far enough. Indeed, the equation might not even have a root in floating point arithmetic — remember that it is a notion which really assumes the continuity and completeness of the number system.

Terminating the search The root search will have to terminate at some stage: the iteration can't go on forever. So we have to decide on some criterion, or tolerance, by which the current value for the root can be assessed. Thus we might require $|x_i - x_{i+1}| < \delta$ where δ has been chosen for some reason related to the physics (for example, 0.01 eV might be considered good enough in this case). Or we can choose a relative value $|(x_i - x_{i+1})/x_i| < \delta$ (provided that we are sure x_i is not zero). In general, the choice we make will depend on the circumstances, and may require experimentation and numerical analysis.

Bracketing the root How do we determine if two values bracket the root? We can write a suitable, slightly complicated, conditional statement to do the obvious tests. A simpler alternative is to search for two values of x, say x_i and x_{i+1}, such that

$$f(x_i)\, f(x_{i+1}) < 0$$

Root Searching: More Advanced Methods

The above methods can be made to work, but they are not necessarily efficient. They may require quite a few evaluations of $f(x)$ in order to converge to a suitable value. Remember that this evaluation will require, for other potentials, a numerical integration of the Schrödinger equation, so it's useful to examine some standard, more efficient, methods.

Newton-Raphson Method This is faster than the above, but more tricky. It is supposed that we know (or can evaluate) the derivative of $f(x)$ at $x = x_i$. Then we can use a tangent at this point to estimate where the root might be. (Draw a diagram, and find out where the tangent intercepts the $x-$axis).

Thus, the new estimate for the position of the root is

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

If $f(x_{i+1}) \neq 0$, then iterate to obtain a succession of points $x_i, x_{i+1}, \ldots$ which approach the roots.

Note the f' in the denominator. If we approach a maximum or minimum turning point of the function $f(x)$ then we can take a large jump ... and end up rather far from where we want to be. Thus, caution is required with these methods.

Secant and False Position Methods These are variants of the Newton-Raphson method, in which we approximate the derivative f' numerically, using a simple first order difference. Then

$$x_{i+1} = x_i - f(x_i)\frac{x_i - x_{i-1}}{f(x_i) - f(x_{i-1})}$$

$$\left(= \frac{f(x_i)x_{i-1} - f(x_{i-1})x_i}{f(x_i) - f(x_{i-1})} \right)$$

For these methods we need two starting values (which may be supplied by a brute-force method) and extrapolate to a third point. One of the original points is then replaced with the new value, and the process is iterated until convergence.

The two methods differ in the way the update is done. In the *Secant* method, the oldest value is discarded. In the *False position* (*regula falsi*) method, the root is bracketed and the appropriate value is updated to maintain this condition.

5.4 Submission

Submit responses and suitable explanations for all of the tasks in Sect. 5.2. Include printouts of all code written.

5.5 Python

We frequently need to compute a new set of values, and then replace the old set with the new set. Python has a useful *tuple assignment* statement that makes this very clean: we don't have to use temporary values to save things that appear on both sides.

For example we can swap two values:

```
x1,x2 = x2,x1  # swap x1, x2
```

or have a more complicated shuffle:

```
x1,f0,f1 = x2,f1,f2  # replace x1 with x2, f0 with f1, f1 with f2
```

In some other language we will probably do this similarly to:

```
temp=x2   # save x2 temporarily
x2=x1     # replace x2
x1=temp   # x1=(old)x2
```