PHYS580 Lab13 Report

Yicheng Feng
PUID: 0030193826

November 24, 2019

Workflow: I use the Linux system, and code C++ in terminals. For visualization, I use the C++ package – ROOT (made by CERN) to make plots. At the end, the reports are written in LaTeX.

The codes for this lab are written as the following files:

- `square_well_1d.h` and `square_well_1d.cxx` for the class `SquareWell1D` to use the shoot method to calculate the wave function in a square well potention (with barrier).
- `lennard_jones_1d.h` and `lennard_jones_1d.cxx` for the class `LennardJones1D` to calculate the wave function in the Lennard-Jones potention with match or variation method.
- `lab13.cxx` for the main function to make plots.

To each problem of this lab report, I will attach the relevant parts of the code. If you want to check the validation of my code, you need to download the whole code from the link `https://github.com/YichengFeng/phys580/tree/master/lab13`.

(1) Use the starter programs `shoot.m` and `potential.m` (or your own equivalent routines) to study one-dimensional quantum mechanics with square-well potentials. First, obtain the first few even and odd parity solutions for a single square well of depth $V = 10^5$. Automate the iteration process so that your program automatically zooms in to the solutions without your having to direct by hand the iteration direction and step size at each step. Then, repeat the calculation with a secondary barrier of height $V = 100$ added at the center of the well, spanning $[-a, a]$ in x. Compare the resulting states and energies to the exact results for infinite barriers given in Eqs. (10.6) (10.8) in the textbook, and explain the differences.

Physics explanation:

We use the shoot method here. We use the binary search to find the wave functions with $\psi(1) = \psi(-1) = 0$:

(a) The energy E_0 is initialized to a value close to the expected one, and ΔE_0 is set to be 0.01.

(b) For the i step, the energy is updated to be $E_i = E_{i-1} + \Delta E_i$. Accordingly, we can calculate $\psi_i(1)$.

- If $\psi_i(1) = 0$, we stop the search and output the results.
- If $\psi_i(1) > \psi_{i-1}(1) > 0$ or $\psi_i(1) < \psi_{i-1}(1) < 0$, we are searching in a wrong direction, so we let $\Delta E_{i+1} = -\Delta E_i$. Then, we go to the beginning of (b).
- If $\psi_i(1)\psi_{i-1}(1) < 0$, we should change direction and halve the step $\Delta E_{i+1} = -0.5\Delta E_i$. Then, we go to the beginning of (b).
- For all other cases, we keep $\Delta E_{i+1} = \Delta E_i$ and go to the beginning of (b).

(c) We stop search and output the results when the current $|\Delta E|$ is less than a certain value e.

In this problem, we set $e = 10^{-10}$.

Figure 1 shows the first three states in the 1D deep square well without barrier: the left pad has even parity and $E = 1.2325$; the middle pad has odd parity and $E = 4.9348$; the right pad has even parity and $E = 11.0922$. The theoretical energy of an infinitely deep 1D square well is

$$E = \frac{\pi^2 n^2 \hbar^2}{8mL^2} = \frac{\pi^2 n^2}{8}, \tag{1}$$

so we should have $E_1 = 1.2237$, $E_2 = 4.9348$, $E_3 = 11.1033$. The simulation and theoretical results are very close. The slight difference is due to the deep but still finite well wall.

Figure 2 shows the first three states in the 1D deep square well with barrier $V_2 = 100$ at $[-0.1, 0.1]$: the left pad has even parity and $E = 5.1398$; the middle pad has odd parity and $E = 5.3251$; the right pad has even parity and $E = 20.4194$. Now the energy is always higher then that without barrier repectively, because the barrier needs extra energy. The states $n = 2i - 1$ and $n = 2i$ has similar energy. The previous even wave function has maximum around $x = 0$, while the previous odd one has 0 there, so the barrier affects more on the even wave functions than the odd ones. As a result, the energy difference between them becomes smaller.

Plots:

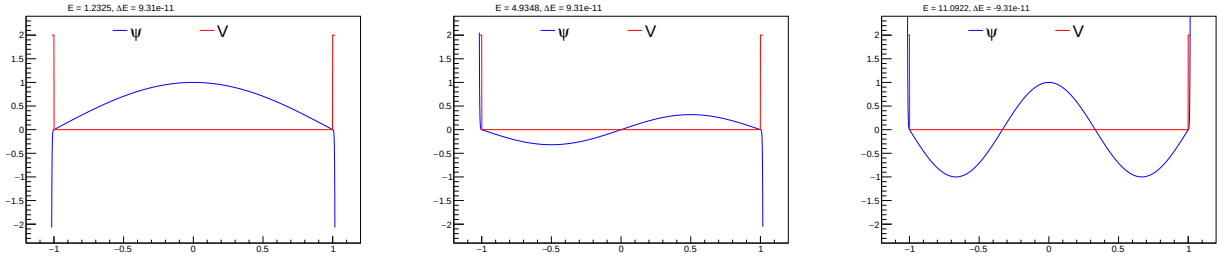

Figure 1: The wave functions (ψ) in the deep square well ($V = 10^5$) depend on various energy: left $E = 1.2325$, middle $E = 4.9348$, right $E = 11.0922$.

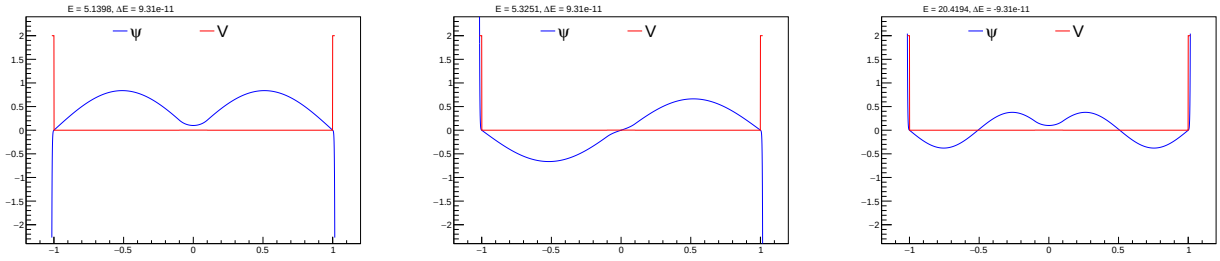

Figure 2: The wave functions (ψ) in the deep square well ($V = 10^5$) depend on various energy: left $E = 5.1398$, middle $E = 5.3251$, right $E = 20.4194$. There is a barrier at $[-0.1, 0.1]$ with $V_2 = 100$.

Relevant code:

For the potential

```
//------------------------------------------------------------------------//

double SquareWell1D::potential(double x) {

        if(fabs(x)<=_a) {
                return _V2;
        } else if(fabs(x)<_L) {
                return 0;
        } else {
                return _Vmax;
        }
}

//------------------------------------------------------------------------//
```

For the one-time shoot calculation

```
//------------------------------------------------------------------------//

void SquareWell1D::cal_once() {

        if(!check()) return;

        _xL.clear();
        _xR.clear();
        _psiL.clear();
```

```
        _psiR.clear();
        _x.clear();
        _psi.clear();

        if(_parity==1) {
                if(_a!=0) {
                        _psiR.push_back(0.1);
                        _psiR.push_back(0.1);
                        _psiL.push_back(0.1);
                        _psiL.push_back(0.1);
                } else {
                        _psiR.push_back(1);
                        _psiR.push_back(1);
                        _psiL.push_back(1);
                        _psiL.push_back(1);
                }
        } else {
                _psiR.push_back(-_dx);
                _psiR.push_back(0);
                _psiL.push_back(-_dx);
                _psiL.push_back(0);
        }
        _xR.push_back(-_dx);
        _xR.push_back(0);
        _xL.push_back(_dx);
        _xL.push_back(0);

        int i = 0;
        while(true) {
                i++;
                double x = (i-1)*_dx;
                double tmpR = 2*_psiR[i] - _psiR[i-1] - 2*(_E - potential(x))*_dx*_dx*_psiR[i];
                _xR.push_back(i*_dx);
                _xL.push_back(-i*_dx);
                _psiR.push_back(tmpR);
                _psiL.push_back(_parity*tmpR);

                if(fabs(_psiR[i+1])>_b) break;
        }

        int n = _psiL.size();
        for(int i=0; i<n; i++) {
                _x.push_back(_xL[n-1-i]);
                _psi.push_back(_psiL[n-1-i]);
        }
        for(int i=2; i<n; i++) {
                _x.push_back(_xR[i]);
                _psi.push_back(_psiR[i]);
        }
}

//------------------------------------------------------------------------//
```

For the binary search

```
//------------------------------------------------------------------------//

void SquareWell1D::adjust_once() {

        cal_once();
        double psiAt1Now = _psiR[floor(_L/_dx+1)];
```

```
        if(psiAt1Now==0) {
                _dE = 0;
        } else if(psiAt1Now*_psiAt1<0) {
                _dE = -0.5*_dE;
        } else if(psiAt1Now>0 && psiAt1Now>_psiAt1) {
                _dE = -_dE;
        } else if(psiAt1Now<0 && psiAt1Now<_psiAt1) {
                _dE = -_dE;
        } else {
                _dE = _dE;
        }

        _E += _dE;
        _psiAt1 = psiAt1Now;
}

//------------------------------------------------------------------------//

void SquareWell1D::adjust_until(double ee) {

        int n = 0;
        while(fabs(_dE)>ee) {
                adjust_once();
                if(n == _iter) break;
                n ++;
        }
}

//------------------------------------------------------------------------//
```

(2) Next, use the programs `matchlj.m`, `lj.m` and `var3.m`, `calc_energy_var.m`, `normalize.m` (or your own equivalent codes) to obtain the first few energy levels (both energies and wave functions) for a Lennard-Jones potential of dimensionless energy scale $\epsilon = 10$ and length scale $\sigma = 1$. Do you get essentially identical ground states with the two methods? For the variational approach, it can be important to employ good strategy to speed up convergence by varying the magnitude of wave function updates (variable fr), and possibly having nonzero "temperature" T along the way (even though you are only trying for the ground state). The strategy of using $T > 0$ in the variational approach is the idea behind simulated annealing.

[OPTIONAL] Try to find an excited bound state with the matching method. To support multiple bound states, you will need to make the LJ well much deeper, e.g., $\epsilon = 30$. Note that the larger the energy, the more nodes the wave function has.

Physics explanation:

We use the match method first. As suggested, the parameters are set $\epsilon = 10$, $\sigma = 1$, and $x_m = 1.4$. We use the binary search to find the wave functions with $\delta = \psi'_L(x_m) - \psi'_R(x_m) \to 0$.

(a) The energy E_0 is initialized to a value close to the expected one, and ΔE_0 is set to be 0.01.

(b) For the i step, the energy is updated to be $E_i = E_{i-1} + \Delta E_i$. Accordingly, we can calculate δ_i.

- If $\delta_i = 0$, we stop the search and output the results.
- If $\delta_i > \delta_{i-1} > 0$ or $\delta_i < \delta_{i-1} < 0$, we are searching in a wrong direction, so we let $\Delta E_{i+1} = -\Delta E_i$. Then, we go to the beginning of (b).
- If $\delta_i \delta_{i-1} < 0$, we should change direction and halve the step $\Delta E_{i+1} = -0.5\Delta E_i$. Then, we go to the beginning of (b).
- For all other cases, we keep $\Delta E_{i+1} = \Delta E_i$ and go to the beginning of (b).

(c) We stop search and output the results when the current $|\Delta E|$ is less than a certain value e.

In this problem, we set $e = 10^{-10}$.

Figure 3 shows the wave functions in the Lennard-Jones potential ($\epsilon = 10, \sigma = 1$) with various energy: left $E = -1.8905$; middle $E = 0.3933$; right $E = 1.5976$. In my trials, there is only one bound state with $\epsilon = 10$.

Then, we use the variation method to find the ground state for the same setting.

Figure 4 shows the result after 10000 attempts with $fr = 0.1$ and another 5000 attempts with $fr = 0.01$. The calculated energy is $E = -1.8569$ which is close to the value from the match method (-1.8905), though not exactly same.

Finally, we change the parameter ϵ into $\epsilon = 30$ to find the excited bound state. Figure 5 shows the wave functions in the Lennard-Jones potential ($\epsilon = 10, \sigma = 1$) with various energy: left $E = -13.2829$; middle $E = -0.6021$; right $E = 11.3298$. The left one is the ground state, and the middle one is the excited bound state.

Plots:

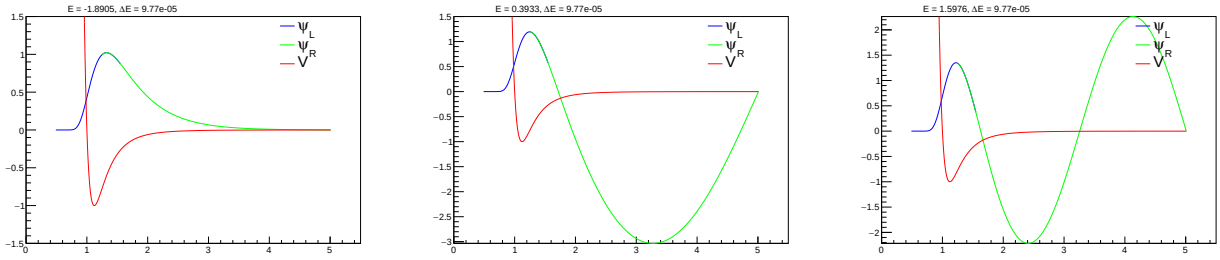

Figure 3: The wave functions in the Lennard-Jones potential ($\epsilon = 10, \sigma = 1$) depend various energy: left $E = -1.8905$; middle $E = 0.3933$; right $E = 1.5976$. The "match method" is used here.

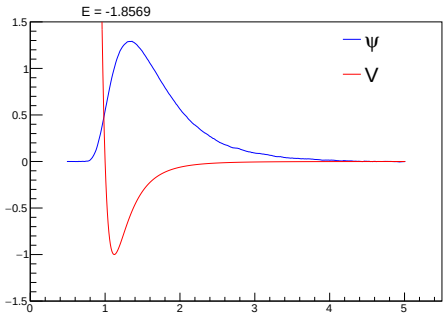

Figure 4: The ground state wave functions is calculated from the variation method with $T = 0$.

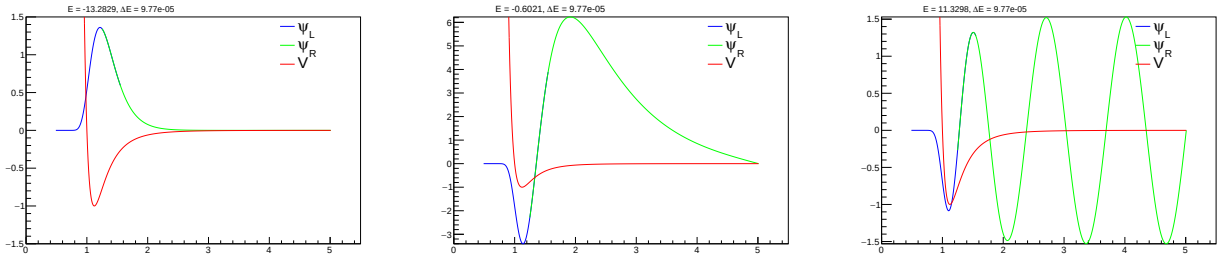

Figure 5: The wave functions in the Lennard-Jones potential ($\epsilon = 30, \sigma = 1$) depend various energy: left $E = -13.2829$; middle $E = -0.6021$; right $E = 11.3298$. The "match method" is used here.

Relevant code:

For the Lennard-Jones potential

```
1 //------------------------------------------------------------------------//
2
3 double LennardJones1D::potential(double x) {
4
5         double invr6 = pow(_sigma/x, 6);
6         double invr12 = invr6*invr6;
7
8         return 4.*_epsilon*(invr12 - invr6);
9 }
```

```
10
11 //------------------------------------------------------------------------//
```

For the match method

```
1  //------------------------------------------------------------------------//
2
3  void LennardJones1D::cal_match_once() {
4
5          if(!check()) return;
6
7          _xL.clear();
8          _xR.clear();
9          _psiL.clear();
10         _psiR.clear();
11         _psi.clear();
12
13         _psiL.push_back(0);
14         _psiL.push_back(1e-2*_dx);
15         _psiR.push_back(0);
16         _psiR.push_back(1e-2*_dx);
17
18         _xL.push_back(_x_left-_dx);
19         _xL.push_back(_x_left);
20         _xR.push_back(_x_right+_dx);
21         _xR.push_back(_x_right);
22
23         double psimatch;
24
25         for(int i=1; i<_matchL+16; i++) {
26                 double x = _x_left + (i-1)*_dx;
27                 double tmpL = 2*_psiL[i] - _psiL[i-1] - 2*(_E - potential(x))*_dx*_dx*_psiL[i];
28                 _psiL.push_back(tmpL);
29                 _xL.push_back(x+_dx);
30         }
31         psimatch = _psiL[_matchL+1];
32         for(int i=0; i<_psiL.size(); i++) {
33                 _psiL[i] /= psimatch;
34         }
35
36         for(int i=1; i<_matchR+16; i++) {
37                 double x = _x_right - (i-1)*_dx;
38                 double tmpR = 2*_psiR[i] - _psiR[i-1] - 2*(_E - potential(x))*_dx*_dx*_psiR[i];
39                 _psiR.push_back(tmpR);
40                 _xR.push_back(x-_dx);
41         }
42         psimatch = _psiR[_matchR+1];
43         for(int i=0; i<_psiR.size(); i++) {
44                 _psiR[i] /= psimatch;
45         }
46 }
47
48 //------------------------------------------------------------------------//
49
50 void LennardJones1D::adjust_match_once() {
51
52         cal_match_once();
53
54         double devL = (_psiL[_matchL+2] - _psiL[_matchL])/(2*_dx);
55         double devR = (_psiR[_matchR] - _psiR[_matchR+2])/(2*_dx);
56         double tmpdiff = devL - devR;
```

```
57
58          if(tmpdiff==0) {
59                  _dE = 0;
60          } else if(tmpdiff*_devdiff<0) {
61                  _dE = -0.5*_dE;
62          } else if(tmpdiff>0 && tmpdiff>_devdiff) {
63                  _dE = -_dE;
64          } else if(tmpdiff<0 && tmpdiff<_devdiff) {
65                  _dE = -_dE;
66          } else {
67                  _dE = _dE;
68          }
69
70          _E += _dE;
71          _devdiff = tmpdiff;
72  }
73
74  //------------------------------------------------------------------------//
75
76  void LennardJones1D::adjust_match_until(double ee) {
77
78          while(fabs(_dE)>ee) {
79                  adjust_match_once();
80          }
81  }
82
83  //------------------------------------------------------------------------//
```

For the variation method

```
1   //------------------------------------------------------------------------//
2
3   bool LennardJones1D::cal_var_once() {
4
5           int n = 1 + floor(1.0*rand()/RAND_MAX*(_iend-2));
6           int m = n + floor(1.0*rand()/RAND_MAX*(_iend-1-n));
7           double p = 1.0*rand()/RAND_MAX;
8           for(int i=n; i<=m; i++) {
9                   _psi_old[i] = _psi[i];
10                  _psi[i] += 2*(p-0.5)*_dpsi;
11          }
12          _newE = var_energy();
13          double deltaE = _newE - _varE;
14          bool keep;
15          if(_newE >= _varE) {
16                  keep = false;
17                  if(_T > 0) {
18                          p = 1.0*rand()/RAND_MAX;
19                          if(p <= exp(-deltaE/_T)) keep = true;
20                  }
21          } else {
22                  keep = true;
23          }
24
25          if(keep) {
26                  _varE = _newE;
27                  var_psi_normalize();
28          } else {
29                  for(int i=n; i<=m; i++) {
30                          _psi[i] = _psi_old[i];
31                  }
```

```
32          }
33
34          return keep;
35  }
36
37  //------------------------------------------------------------------------//
38
39  int LennardJones1D::cal_var_attempts(int n_attempts) {
40
41          int n_move = 0;
42          for(int i_attempts=0; i_attempts<n_attempts; i_attempts++) {
43                  if(cal_var_once()) n_move ++;
44          }
45          _n_total += n_attempts;
46          _n_moves += n_move;
47
48          return n_move;
49  }
50
51  //------------------------------------------------------------------------//
```