Numerical solution of the time-independent 1-D Schrödinger equation

Matthias E. Möbius

September 24, 2010

1 Aim of the computational lab

- Numerical solution of the one-dimensional stationary Schrödinger equation.
- Employ the 3-point Numerov algorithm and shooting/matching method to find the energy Eigenvalues and corresponding Eigenfunctions for the infinite square well.
- Check accuracy by comparing with the analytic solution.
- Find eigenstates of a more complicated potential.
- Check orthogonality of the eigenfunctions using numerical integration.

2 Introduction

The Schrödinger equation lies at the heart of quantum mechanics. In this computational project you will numerically solve for the stationary solutions of the wave equation in one spatial dimension. The time-independent form of the Schrödinger equation is given by:

$$E\psi(x) = -\frac{\hbar^2}{2m}\frac{d^2\psi}{dx^2} + V(x)\psi(x), \tag{1}$$

where $\psi(x)$ is the wavefunction, $V(x)$ the potential energy, m the particle mass, and $\hbar$ is Planck's constant divided by 2π. For simplicity the particle will be bound by infinite wells, such that the potential $V(x = 0)$ and $V(x = L)$ is infinite.

Note that this is an *E*igenvalue equation. Only for certain values of E there will be a solution that satisfies the boundary conditions. Moreover, any solution $\psi(x)$ multiplied by an arbitrary constant is still a solution to this equation.

In order to avoid computations with small numbers such as $\hbar$, we non-dimensionalize the equation:

$$\frac{d^2\psi(\tilde{x})}{d\tilde{x}^2} + \gamma^2\left(\epsilon - \nu(\tilde{x})\right)\psi(\tilde{x}) = 0, \tag{2}$$

where $\nu(\tilde{x}) = V(\tilde{x})/V_0$ is a dimensionless potential energy with a range of values between -1 and $+1$. This can always be achieved by taking V_0 to be the

maximum deviation from zero energy. $\epsilon = E/V_0$ is a dimensionless energy, and

$$\gamma^2 = \frac{2mL^2V_0}{\hbar^2}, \tag{3}$$

where L is the physical size of the well. γ^2 is also dimensionless and depends on the mass m of the particle, the depth V_0 of the well, and the size of the well. Note that the spatial variable is also rescaled $\tilde{x} = x/L$ and goes from 0 (left side of the wall) to +1 (right side of the wall).

We now have to discretise the equation in order to numerically solve it. We divide the spatial variables with a step size $l \equiv 1/(N-1)$, where N is the number of points. There are a number of different integration schemes, but for this type of equation, the so-called Numerov algorithm works very well as it is stable and accurate. For a detailed description consult appendix I.

An equation of the type

$$\frac{d^2\psi}{dx^2} + k^2(x)\psi(x) = 0 \tag{4}$$

can be integrated by the Numerov 3-point formula:

$$\psi_{n+1} = \frac{2\left(1 - \frac{5}{12}l^2k_n^2\right)\psi_n - \left(1 + \frac{1}{12}l^2k_{n-1}^2\right)\psi_{n-1}}{1 + \frac{1}{12}l^2k_{n+1}^2} \tag{5}$$

By specifying two neighbouring points, one can obtain the third point. For the Schrödinger equation,

$$k_n = \gamma^2\left(\epsilon - \nu(x_n)\right) \tag{6}$$

Since the particle is bound by infinite wells, we know that the wavefunction ψ vanishes at both ends of the well and beyond: $\psi(x=0) = \psi(x=+1) = 0$, where +1 corresponds to the right hand side of the well. In order to find the solution numerically, the procedure is as follows. We integrate from both ends of the well, by using the boundary conditions at both ends of the well. Note, that the first point after the boundary, $\psi(x_1)$ and $\psi(x_{N-2})$ is arbitrary, since $\psi(x)$ can always be rescaled by any constant. Even if ψ does not satisfy the boundary conditions, it still satisfies the Schrödinger equation.

3 The source code template

You do not have to start from scratch. You can download a template (and information) from

`http://www.tcd.ie/Physics/People/Matthias.Moebius/teaching/`

The hardcopy can be found in Appendix II of the handout. Read the code carefully. The main program defines the number of points $N = 1000$ and the integration step l. It also sets the dimensionless quantity $\gamma^2 = \frac{2mL^2V_0}{\hbar^2} = 100$ (do not change this). The matching point is set in the middle of the well. First the potential is initialized with the function `InitPotential`. After that, `FindEigenstate` is called. This function integrates the Schröedinger equation from both sides of the well to the matching point `MatchPoint` for a given (dimensionless) energy E and potential. Finally, the program outputs the resulting wave function `Psi` to a text file which can be plotted with `gnuplot` or Excel, for example.

4 Tasks

1. Solve the *non-dimensional* Schrödinger equation (2) analytically for the infinite square well where $V(x) = -V_0$ for $0 < x < L$ and find an expression for the non-dimensional energy Eigenvalues in terms of γ^2.

2. Download the template. The file is called `schrodtemp.cpp` and is printed in the appendix II of this handout. Open a terminal window and compile the code using g++:

   `g++ schrodtemp.cpp -o schrod`.

   After compilation you can run the program by typing `./schrod`. When running the code you will be asked to enter a trial energy (in non-dimensional units, of course). The program will integrate the Schrödinger equation for this particular value from both sides of the well until the matching point, which is chosen to be in the middle of the well. Also, the potential is chosen to be a constant: $v(\tilde{x}) = -1$. The program uses $N = 1000$ data points and γ^2 is set to 100. Keep this values. The program also outputs $\psi(x)$ for the corresponding trial energy into a text file called `Psi.dat`. Plot the wavefunction (e.g. with `gnuplot`) for a number of trial energies above and below the ground state energy you computed in part 1.

3. Open the template in your favourite text editor (`gedit` for example). Read the code carefully and make sure you understand what it is doing. Insert some code into `FindEigenstate` that outputs the difference in the slopes of ψ_{left} (array `PsiLeft`) and ψ_{right} (array `PsiRight`) at the matching point. Use the Euler difference to compute the slopes: $\psi'(x) \approx [\psi(x+l) - \psi(x)]/l$. Verify (manually) that this difference is minimized at the ground state energy by looking at this difference for trial energies above and below the ground state energy.

4. If the trial energy is not an Eigenvalue the slope will be *discontinuous* at the matching point. Improve the code such that the program will search for the Eigenvalue by minimizing the difference in the slopes of the two wave functions (ψ_{left} and ψ_{right}) at the matching point. There are different ways to achieve this. One way is to start with an trial energy E below the Eigenstate. Then you compute $\psi_{left}(x)$ and $\psi_{right}(x)$ for $E + \Delta E$, where ΔE is a suitably chosen energy increment. Compute the difference in slopes at the matching point, $\psi'_{left}(x_{match}) - \psi'_{right}(x_{match})$, for E and $E + \Delta E$. If the difference changed sign, then let $\Delta E = -\Delta E/2$. Iterate until you have the desired accuracy.

5. How accurate can you get the ground state energy eigenvalue? Compare with the analytical solution.

6. This minimization procedure will not work for the first excited state. However, it will work for the second excited state. Why? Can you think of an easy way to to search for the first excited state?

7. In order to find the other (odd) Eigenstates change the code as follows: Choose a different matching point and scale ψ_{right} such that $\psi_{right}(x_{match}) = \psi_{left}(x_{match})$. Recall that $\psi(x)$ can always be multiplied by a constant

and still be a solution to the Schrödinger equation. Try different matching points. This minimization will fail for some matching points, e.g. `MatchPoint=N/2` for the first excited state. Why?

8. Find the first 6 energy Eigenstates and verify that they agree with the analytical solution.

9. The Eigenfunctions are not normalized yet. Write a function that normalizes $\psi(x)$. You may use Simpson's rule for numerical integration:

$$\begin{aligned}\int_{x_0}^{x_N} f(x)dx &\simeq \frac{l}{3}\left(f_0 + 4f_1 + 2f_2 + 4f_3 + 2f_4 + \ldots + 2f_{N-2} + 4f_{N-1} + f_N\right) \\ &= \frac{l}{3}\left(f_0 + 2\sum_{j=1}^{N/2-1} f_{2j} + 4\sum_{j=1}^{N/2} f_{2j-1} + f_N\right)\end{aligned}$$

   Compare your normalized Eigenfunctions with the analytical solution.

10. Change the potential in the function `InitPotential`. Comment out the infinite square potential and uncomment the step potential just below. The potential $\nu(x) = -1$ for $0 < \tilde{x} < 0.5$ and 0 for $0.5 < \tilde{x} < 1$. Find the ground state of the system. How many Eigenstates have energies below 0? NOTE: It is important to choose the matching point to be outside the bump! The reason is to avoid numerically integrating *into* a classically forbidden region. Can you think of a reason why? (Hint: What functional form does $\psi(x)$ have in the classically forbidden region? Are all these solutions physical?)

11. Compute the first 6 eigenstates for this potential. What is the probability for the particle to be in the right half of the well where the bump is located for these six eigenstates? What is the asymptotic value of this probability as you go to higher energies?

12. Check orthogonality for some of these Eigenfunctions.

13. Try out the linear potential in `InitPotential` and find the first 6 eigenstates.

5 Appendix I

5.1 The Numerov algorithm

The Numerov algorithm is a popular numerical method to integrate equations of the form:

$$\frac{d^2\psi}{dx^2} + k^2(x)\psi(x) = 0, \tag{7}$$

of which the stationary Schrödinger equation is one example. In principle this could be solved via a 4^{th} order Runge-Kutta. However, the Numerov algorithm takes advantage of the particular structure of the equation, which involves no first derivative and is linear in ψ. In the following, we derive the Numerov integration scheme for the above equation:

The Taylor expansion of $\psi(x+l)$ gives

$$\psi(x+l) = \psi(x) + l\psi'(x) + \frac{l^2}{2}\psi^{(2)}(x) + \frac{l^3}{6}\psi^{(3)}(x) + \frac{l^4}{24}\psi^{(4)}(x) + \dots \tag{8}$$

Adding this to the series expansion of $\psi(x-l)$ all the odd powers of l vanish:

$$\psi(x+l) + \psi(x-l) = 2\psi(x) + l^2\psi^{(2)}(x) + \frac{l^4}{12}\psi^{(4)}(x) + O(l^6) \tag{9}$$

Solving for the second derivative we obtain

$$\psi^{(2)}(x) = \frac{\psi(x+l) + \psi(x-l) - 2\psi(x)}{l^2} - \frac{l^2}{12}\psi^{(4)}(x) + O(l^4) \tag{10}$$

In order to evaluate the 4^{th} order derivative that appears on the RHS, we apply $1 + (l^2/12)d^2/dx^2$ to eqn.7:

$$\psi^{(2)}(x) + \frac{l^2}{12}\psi^{(4)}(x) + k^2(x)\psi(x) + \frac{l^2}{12}\frac{d^2}{dx^2}[k^2(x)\psi(x)] = 0 \tag{11}$$

Substituting $\psi^{(2)}(x) + \frac{l^2}{12}\psi^{(4)}(x)$ form eqn. 11 into eqn. 10 we obtain

$$\psi(x+l)+\psi(x-l)-2\psi(x)+l^2k^2(x)\psi(x)+\frac{l^4}{12}\frac{d^2}{dx^2}[k^2(x)\psi(x)]+O(l^6) = 0 \tag{12}$$

By applying the centered difference twice we can evaluate the last term:

$$\frac{d^2}{dx^2}[k^2(x)\psi(x)] \approx \frac{k^2(x+l)\psi(x+l) + k^2(x-l)\psi(x-l) - 2k^2(x)\psi(x)}{l^2} \tag{13}$$

Combining eqns. 12 and 13 and solving for $\psi(x+h)$ we get

$$\psi(x+l) = \frac{2\left(1 - \frac{5}{12}l^2k^2(x)\right)\psi(x) - \left(1 + \frac{1}{12}l^2k^2(x-l)\right)\psi(x-l)}{1 + \frac{1}{12}l^2k^2(x+l)} \tag{14}$$

We rewrite the above expression by setting $x = x_n \equiv x_0 + nl$ and $k_n \equiv k(x_n)$. Then we obtain

$$\psi_{n+1} = \frac{2\left(1 - \frac{5}{12}l^2k_n^2\right)\psi_n - \left(1 + \frac{1}{12}l^2k_{n-1}^2\right)\psi_{n-1}}{1 + \frac{1}{12}l^2k_{n+1}^2} \tag{15}$$

Even though the error at each integration step is only $O(l^6)$, it accumulates over many steps and the accuracy turns out to be of order $O(l^4)$, which is still quite good.

6 Appendix II

6.1 Source code of schrodsolve.cpp

see next page.

```
////////////////////////////////////////////////////////////////
//                                                            //
// Solutions to the 1D time-independent Schrodinger equation  //
//                                                            //
//  Template by M.Moebius (2010)                              //
//                                                            //
////////////////////////////////////////////////////////////////


// header files
# include <iostream> // input output streams
# include <fstream>  // file input output streams
# include <iomanip>  // manipulation of input and output streams
# include <stdlib.h> // standard library
# include <cmath>    // math functions

using namespace std;

//
// function prototypes
//

void Numerov(double *PsiLeft,double *PsiRight, int N, double *KSquared, double l, int
    MatchPoint);
void UpdateKSquared(double *KSqared, double *Potential, double gamma_sq, double E, int N);
void InitPotential(double *Potential, int N);
void FindEigenstate(double& E, int N, double *KSquared, double l, int TurnPoint, double *
    Psi, double gamma_sq);
void OutputData(double *Psi, int N);


// main program
int main()
{
    // variable declarations

    int N;                  // number of space points
    N=1000;
    double l;               // step size = 1/(N-1)
    l=1.0/(N-1);
    double E;               // Energy
    double gamma_sq;        // Dimensionless parameter = 2*mass*L^2*V0/hbar^2
    int MatchPoint;         // The point at which the left and right wavefunctions are
    matched

    double *Psi;
    double *Potential;
    Psi = new double [N];           // Wavefunction
    Potential = new double [N];     // Potential energy inside the well


    // initialize parameters

    gamma_sq=100.0;                 // gamma includes all the relevant physical constants
    MatchPoint=N/2;                 // Matchpoint (=N/2 corresponds to the middle of the
    well)

    cout << "Trial Energy: ";       // ask user for trial energy
    cin >> E;

    InitPotential(Potential,N);     // initialize potential within the infinite well

    // Find Eigenstate
    // in this template, the function FindEigenstate only computes Psi for the trial energy

    FindEigenstate(E, N, Potential, l, MatchPoint, Psi, gamma_sq);

    // output the wavefunction to a ASCII file named Psi.dat
    OutputData(Psi,N);

    // Output the corresponding energy to the console (12 digit precision after the decimal
     point)
    cout << setiosflags(ios::fixed) << setprecision(12)<< "Energy: " << E;
```

```
    // free the memory
    delete[] Psi;
    delete[] Potential;

}
// this function computes Psi for a given energy
//
void FindEigenstate(double& E, int N, double *Potential, double l, int MatchPoint, double *
    Psi, double gamma_sq)
{
    int i; // dummy variable
    double *PsiLeft;
    double *PsiRight;
    double *KSquared;

    PsiLeft = new double [N];
    PsiRight = new double [N];
    KSquared = new double [N];

    // update KSquared for trial energy E
    UpdateKSquared(KSquared, Potential, gamma_sq, E, N);
    // Integrate from both sides to the matching point
    Numerov(PsiLeft, PsiRight, N, KSquared, l, MatchPoint);

    ////////////////////////////////////////////////////
    //
    // You should put your slope matching procedure here
    //
    ////////////////////////////////////////////////////

    // merge the left and right integration into Psi
    for (i=0;i<N;i++)
    {
        if (i<MatchPoint)
        {
            Psi[i]=PsiLeft[i];
        } else
        {
            Psi[i]=PsiRight[i];
        }
    }

    delete[] PsiRight;
    delete[] PsiLeft;
    delete[] KSquared;
}

//
// this function performs the numerical integration from both sides of the well
//
void Numerov(double *PsiLeft,double *PsiRight, int N, double *KSquared, double l, int
    MatchPoint)
{
    int i;              //dummy variable
    double constant;    // =l^2/12

    constant=l*l/12;

    PsiLeft[0]=0;                   // Initialize left boundary
    PsiLeft[1]=0.0001;

    PsiRight[N-1]=0;
    PsiRight[N-2]=0.0001;           // Initialize right boundary

    // Shoot from the left to the Matchpoint

    for (i=2;i<(MatchPoint+1);i++)
    {
        PsiLeft[i]=(2.0*(1.0-(5.0*constant*KSquared[i-1]))*PsiLeft[i-1]-(1.0+constant*
    KSquared[i-2])*PsiLeft[i-2])/(1.0+constant*KSquared[i]);
    }

    // Shoot from the right to the (Matchpoint-1)
```

```
    for (i=N-3;i>(MatchPoint-2);i--)
    {
        PsiRight[i]=(2.0*(1.0-(5.0*constant*KSquared[i+1]))*PsiRight[i+1]-(1.0+constant*
    KSquared[i+2])*PsiRight[i+2])/(1.0+constant*KSquared[i]);
    }

}
//
// this function is called at the beginning to initialize the potential energy v(x)
//
void InitPotential(double *Potential,int N)
{
    int i;  // dummy variable
    for (i=0;i<N;i++)
    {
        // Infinite square well
         Potential[i]=-1.0;

        // Step Potential
        /*if (i > N/2) {
            Potential[i]=0;
        } else {
            Potential[i]=-1;
        }*/

        // linear Potential
        // Potential[i]=2*float(i)/(N-1)-1;
    }

}
//
// this function computes KSquared for a given trial energy E
//
void UpdateKSquared(double *KSquared, double *Potential, double gamma_sq, double E, int N)
{
    int i;      // dummy variable
    for (i=0;i<N;i++)
    {
        KSquared[i]=gamma_sq*(E-Potential[i]);
    }
}
//
// this function takes a vector of size N and outputs it to an ASCII file named Psi.dat
//
void OutputData(double *Data, int N)
{
    int i;
    fstream Datafile("Psi.dat",ios::out);

    for (i=0;i<N;i++)
    {
        Datafile << Data[i] << "\n";
    }
    Datafile.close();                       // close file
}
```