Harmonic Oscillator: Numerov Algorithm

Linear equations and the Sturm-Liouville problem

Many important differential equations in physics are second order and *linear* in the solution $u(x)$ of the form

$$\frac{\mathrm{d}^2u}{\mathrm{d}x^2} + d(x)\frac{\mathrm{d}u}{\mathrm{d}x} + q(x)u = s(x) \ , \tag{1}$$

where $d(x)$, $q(x)$ and $s(x)$ are given functions. The *source* function $s(x)$ makes the equation *inhomogenous*. If $s(x) = 0$ the equation is *homogeneous*.

The *Sturm-Liouville* problem deals with linear homogeneous second order equations of the form

$$-\frac{\mathrm{d}}{\mathrm{d}x}\left[p(x)\frac{\mathrm{d}u}{\mathrm{d}x}\right] + r(x)u(x) = \lambda w(x)u(x) \ , \tag{2}$$

where $p(x)$, $r(x)$ and $w(x)$ are given functions, and λ is a parameter whose value will be determined by solving the equation. The coefficient function $p(x)$ and the *weight function* $w(x)$ are assumed to be positive in the integration domain $a \leq x \leq b$.

To obtain a definite solution, boundary conditions must be imposed. Sturm and Liuoville considered homogeneous and linear boundary conditions of the following general type

$$c_1u(a) + c_2u'(a) = 0 \ , \qquad c_3u(a) + c_4u'(a) = 0 \ , \tag{3}$$

where c_i are given constants which do not depend on the parameter λ.

The regular Sturm Liouville problem

The Sturm-Liouville system has a *trivial* solution $u(x) = 0$ which obviously satisfies the boundary conditions as well! Sturm and Liouville showed that this general system has many remarkable properties, some of which are:

- Non-trivial solutions exist only for a discrete set of real values of the parameter λ, which are the *eigenvalues* of the system.
- To each eigenvalue corresponds a unique *eigenfunction*.
- The eigenfunctions are *orthogonal* with weight $w(x)$.
- If the eigenvalues are arranged in increasing order, then successive eigenfunctions each have one additional node or zero in the integration domain.

These properties are very important for applications in quantum mechanics and other subjects.

The Numerov Algorithm

This is a very efficient algorithm[1] for solving second order linear differential equations, including Sturm-Liouville equations.

Consider the equation

$$\frac{\mathrm{d}^2u}{\mathrm{d}x^2} + q(x)u(x) = s(x) \ , \tag{4}$$

which includes many interesting applications. Note that the equation does not have a first-order derivative term.

The symmetric three-point difference formula for the second derivative is

$$\frac{u_{n+1} + u_{n-1} - 2u_n}{h^2} = u_n'' + \frac{h^2}{12}u_n'''' + \mathcal{O}(h^4) \ . \tag{5}$$

Using the differential equation, we can write

$$u_n'''' = \frac{\mathrm{d}^2}{\mathrm{d}x^2}\left(-q(x)u(x) + s(x)\right)\Big|_{x=x_n} \tag{6}$$

$$= -\frac{q_{n+1}u_{n+1} - 2q_nu_n + q_{n-1}u_{n-1}}{h^2} + \frac{s_{n+1} - 2s_n + s_{n-1}}{h^2} + \mathcal{O}(h^2) \ . \tag{7}$$

Plugging this into the difference formula and collecting terms gives

$$\left(1 + \frac{h^2}{12}q_{n+1}\right)u_{n+1} - 2\left(1 + \frac{5h^2}{12}q_n\right)u_n + \left(1 + \frac{h^2}{12}q_{n-1}\right)u_{n-1} \tag{8}$$

$$= \frac{h^2}{12}\left(s_{n+1} + 10s_n + s_{n-1}\right) + \mathcal{O}(h^6) \ . \tag{9}$$

The local error is one order in h more accurate than fourth-order Runge Kutta! Because of its compact and efficient form, and its high accuracy, the Numerov algorithm was very popular in the early 20-th century for doing atomic structure calculations soon after the discovery of quantum mechanics.

Note that the Numerov algorithm is a three-point formula: to start the algorithm, two initial values, u_0 and u_1, are needed.

Time-independent Schrödinger Equation

Quantum mechanics is more complicated than classical mechanics. The trajectory of a classical particle moving in one dimension can be described in a two-dimensional phase space. To describe the state of a quantum particle moving in one dimension we use a *Hilbert space*: this space is defined on *complex numbers* instead of real; and it has an infinite number of dimensions!

The state vector is determined by solving a partial differential equation

$$i\hbar\frac{\partial\psi}{\partial t} = -\frac{\hbar^2}{2m}\frac{\partial^2\psi}{\partial x^2} + V(x)\psi(x,t) \ . \tag{10}$$

A big simplification occurs if we wish to solve for energy eigenstates:

$$\psi(x,t) = e^{-iEt/\hbar}\phi(x) \ , \tag{11}$$

which are described by eigenfunctions $\phi(x)$ of the Hamiltonian

$$H\phi(x) = \left[-\frac{\hbar^2}{2m}\frac{\partial^2}{\partial x^2} + V(x)\right]\phi(x) = E\phi(x) \ . \tag{12}$$

This is a second order ordinary differential equation like Newton's equation equation of motion. However, it is more complicated to solve because physically sensible (normalizable) solutions exist only for very special values of the energy E.

Equation and normalizable solutions

We would like to find bound state solutions of

$$\frac{d^2\phi}{dx^2} = \frac{2m}{\hbar^2}[V(x) - E]\phi(x) \ , \tag{13}$$

for example for the Harmonic oscillator

$$V(x) = \frac{1}{2}m\omega^2 x^2 \ . \tag{14}$$

These are *normalizable* solutions

$$\int_{-\infty}^{\infty} dx|\phi(x)|^2 = \text{a finite positive number}, \tag{15}$$

which implies that $\phi(|x| \to \infty)$ tends to zero sufficiently rapidly. Such solutions exist only for discrete eigenvalues of the energy parameter E. For the harmonic oscillator, $E_n = (n + \frac{1}{2})\hbar\omega$ where $n = 0, 1, 2, \ldots$

Boundary conditions

It is not practical in a numerical program to impose boundary conditions at $x = \pm\infty$. We need to choose a finite interval $x_{\text{left}} \le x \le x_{\text{right}}$ and apply approximate boundary conditions at the left and right boundaries.

For a bound state problem, the particle is confined mostly in the *classically allowed region* $E > V(x)$, and $\phi(x)$ decays rapidly to zero in the *classically forbidden regions* $V(x) > E$.

For the harmonic oscillator, the allowed region is near the origin $|x| < \sqrt{2E/m}/\omega$. It makes sense to choose $x_{\text{left}} \ll -\sqrt{2E/m}/\omega$ and $x_{\text{right}} \gg \sqrt{2E/m}/\omega$. As approximate boundary conditions we can set $\phi(x_{\text{left}}) = 0$ and $\phi(x_{\text{right}}) = 0$. These would be the exact boundary conditions for a modified potential $V(x < x_{\text{left}}) = \infty$ and $V(x > x_{\text{right}}) = \infty$.

Integrating the equation

To use a marching algorithm, such as Numerov's, we can choose a step size h and set up a lattice of points between x_{left} and x_{right}. If we start from the left, we need to specify the values of $\phi(x)$ at the *first two* lattice points. We set $\phi(x_{\text{left}}) = 0$ to approximate the physical boundary conditions.

But what about $\phi(x_{\text{left}} + h)$? Actually, if $\phi(x_{\text{left}}) = 0$ then $\phi(x_{\text{left}} + h)$ can be set to any arbitrary constant! It is easy to see that this constant will affect only the normalization and not the x-dependence of the solution. This is because the Schrödinger equation, and the Numerov recursion formula, are linear and homogeneous in ϕ. Multiplying the solution by a constant also solves the equation.

Instabilities and need for a matching point

One might be tempted to say: prime the Numerov algorithm at x_{left} and march to x_{right} and check whether $\phi(x_{\text{right}}) = 0$. If it is non-zero, adjust the energy E until it is zero.

Unfortunately, this does not work! The reason is that there are two solutions the forbidden region: the physical solution decays rapidly to zero, but there is an unphysical solution which grows exponentially to $\pm\infty$. Even if you are lucky enough to choose E to be exactly equal to an energy eigenvalue, numerical errors will cause the solution to run away.

The solution to this instability is to generate *two* solutions: starting at $x_{\rm left}$, generate $\phi_{\rm left}$ with initial values $\phi(x_{\rm left}) = 0$ and $\phi(x_{\rm left} + h) = 10^{-10}$ by integrating toward $x = 0$, and starting at $x_{\rm right}$, generate $\phi_{\rm right}$ with initial values $\phi(x_{\rm right}) = 0$ and $\phi(x_{\rm right} - h) = 10^{-10}$ by integrating backwards toward $x = 0$. If E is an eigenvalue, then these two solutions can be made to match exactly.

To check whether we have a match we can choose a matching point $x_{\rm match}$ somewhere in the allowed region. By multiplying one of the solutions by a constant we can always ensure that

$$\phi_{\rm left}(x_{\rm match}) = \phi_{\rm right}(x_{\rm match}) \ . \tag{16}$$

The test for a true match is that

$$\left.\frac{d\phi_{\rm left}}{dx}\right|_{x_{\rm match}} = \left.\frac{d\phi_{\rm right}}{dx}\right|_{x_{\rm match}} \ . \tag{17}$$

Matching at a turning point

One possibility is to choose the matching point at one of the *classical turning points* of the potential, say the right turning point. Recall from classial mechanics that a turning point for a particle with energy E occurs when $E = V(x)$.

We will use the following matching condition:

$$\begin{aligned} F(E) &= \pm\left[\phi_{\rm left}(x_{\rm match} + h) - \phi_{\rm left}(x_{\rm match} - h)\right] && (18)\\ &\quad - \left[\phi_{\rm right}(x_{\rm match} + h) - \phi_{\rm right}(x_{\rm match} - h)\right] / (2h\phi(x_{\rm match})) && (19)\\ &= 0 \ . && (20) \end{aligned}$$

The numerator divided by $2h$ is just the difference of the three-point formulas for the derivatives of $\phi_{\rm left}$ and $\phi_{\rm right}$. Dividing by $\phi(x_{\rm match}) = \phi_{\rm left}(x_{\rm match}) = \phi_{\rm right}(x_{\rm match})$ ensures that the matching function is independent of the normalizations of the left and right solutions. The $\pm$ sign will be adjusted so that $F(E)$ is a *continuous* function of E. This helps in searching systematically for eigenvalues.

So the basic strategy for finding the eigenvalues E_n is to compute the *matching function* $F(E)$ numerically and then find its *roots*, i.e., the values of E at which $F(E) = 0$.

C++ Program to Find Harmonic Oscillator Eigenstates

The following program implements the strategy outlined above.

Program 1: `http://www.physics.buffalo.edu/phy410-505/topic5/schroedinger.cpp`

```
#include <cmath>
#include <cstdlib>
#include <iostream>
#include <iomanip>
#include <fstream>
using namespace std;

#include "linalg.hpp"
#include "findroot.hpp"
using namespace cpl;
```

Define the potential for the harmonic oscillator

$$V(x) = \frac{1}{2} m\omega^2 x^2 \ , \tag{21}$$

and the Sturm-Liouville $q(x)$ function:

Program 1: http://www.physics.buffalo.edu/phy410-505/topic5/schroedinger.cpp

```
const double hbar = 1;              // Planck's constant / 2pi
const double m = 1;                 // particle mass
double omega = 1;                   // oscillator frequency

double V(double x) {                // harmonic oscillator potential
  return 0.5 * m * omega * omega * x * x;
}

double E;                           // current energy in search

double q(double x) {                // Sturm-Liouville q function
  return 2 * m / (hbar * hbar) * (E - V(x));
}
```

Set up a grid of $N + 1$ points and grid spacing $h = (x_{\rm right} - x_{\rm left})/N$.

Program 1: http://www.physics.buffalo.edu/phy410-505/topic5/schroedinger.cpp

```
int N = 500;                        // number of lattice points = N + 1
double x_left = -5;                 // left boundary
double x_right = 5;                 // right boundary
double h = (x_right - x_left) / N; // grid spacing
```

Allocate vectors for $\phi_{\rm left}$ and $\phi_{\rm right}$, and also a vector for the whole solution:

Program 1: http://www.physics.buffalo.edu/phy410-505/topic5/schroedinger.cpp

```
vector<double> phi_left(N+1);       // wave function integrating from left
vector<double> phi_right(N+1);      // wave function integrating from right
vector<double> phi(N+1);            // whole wavefunction
```

Now define the search function $F(E)$.

Program 1: http://www.physics.buffalo.edu/phy410-505/topic5/schroedinger.cpp

```
double F(double E) {                // eigenvalues at F(E) = 0
```

```
// set global E value needed by the q(x) function
::E = E;

// find right turning point
int i_match = N;
double x = x_right;             // start at right boundary
while (V(x) > E)  {             // in forbidden region
  --i_match;
  x -= h;
  if (i_match < 0) {
    cerr << "can't find right turning point" << endl;
    exit(EXIT_FAILURE);
  }
}
```

We will use the Numerov algorithm to perform the integrations.

Program 1: `http://www.physics.buffalo.edu/phy410-505/topic5/schroedinger.cpp`

```
// integrate phi_left using Numerov algorithm
phi_left[0] = 0;
phi_left[1] = 1e-10;
double c = h * h / 12;          // constant in Numerov formula
for (int i = 1; i <= i_match; i++) {
  x = x_left + i * h;
  phi_left[i+1]  = 2 * (1 - 5 * c * q(x)) * phi_left[i];
  phi_left[i+1] -= (1 + c * q(x - h)) * phi_left[i-1];
  phi_left[i+1] /= 1 + c * q(x + h);
}

// integrate phi_right
phi[N]   = phi_right[N]   = 0;
phi[N-1] = phi_right[N-1] = 1e-10;
for (int i = N - 1; i >= i_match; i--) {
  x = x_right - i * h;
  phi_right[i-1]  = 2 * (1 - 5 * c * q(x)) * phi_right[i];
  phi_right[i-1] -= (1 + c * q(x + h)) * phi_right[i+1];
  phi[i-1] = phi_right[i-1] /= 1 + c * q(x - h);
}
```

Now we need to impose the condition $\phi_{\rm left}(x_{\rm match}) = \phi_{\rm right}(x_{\rm match})$. This is easily done by multiplying $\phi_{\rm left}$ by the appropriate constant.

Program 1: `http://www.physics.buffalo.edu/phy410-505/topic5/schroedinger.cpp`

```
// rescale phi_left
```

```
double scale = phi_right[i_match] / phi_left[i_match];
for (int i = 0; i <= i_match + 1; i++)
  phi[i] = phi_left[i] *= scale;
```

Next we need to fix the sign of the seach function so it is continuous. If this is not done, the search function will change sign whenever the wavefunction develops a new node as shown in Fig. 1. Root finding routines which assume that a change in sign of the function signals the presence of a root will be confused by this discontinuity.

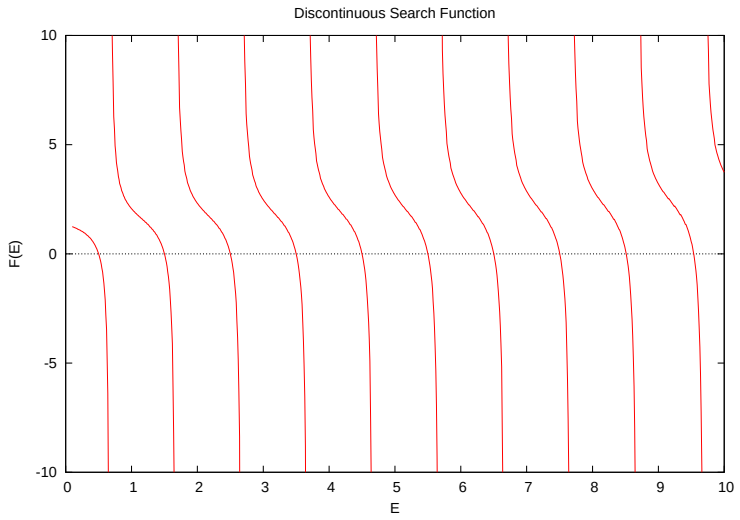

Figure 1: Search function with discontinuities.

We define `static` variables to keep track of the current sign used and the current number of nodes. The values of `static` variables are preserved from one call of the function to the next.

Program 1: `http://www.physics.buffalo.edu/phy410-505/topic5/schroedinger.cpp`

```
// make F(E) continuous
static int sign = 1;            // current sign used
static int nodes = 0;           // current number of nodes

// count number of nodes in phi_left
int n = 0;
for (int i = 1; i <= i_match; i++)
  if (phi_left[i-1] * phi_left[i] < 0)
    ++n;

// flip its sign when a new node develops
if (n != nodes) {
  nodes = n;
  sign = -sign;
}
```

Including this sign in $F(E)$ results in a continuous function as shown in Fig. 2.

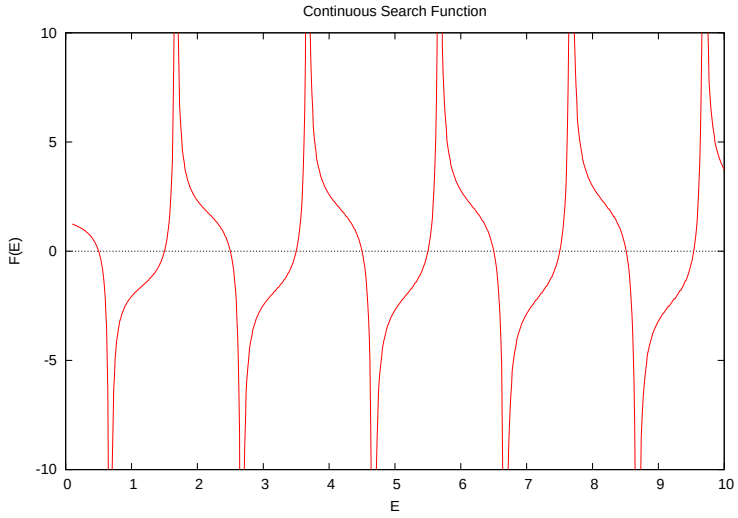

Figure 2: Search function made continuous.

Program 1: `http://www.physics.buffalo.edu/phy410-505/topic5/schroedinger.cpp`

```
  return sign * ( phi_right[i_match-1] - phi_right[i_match+1]
                  - phi_left[i_match-1] + phi_left[i_match+1] )
         / (2 * h * phi_right[i_match]);
}
```

The following function normalizes the wavefunction so that

$$\int_{x_{\rm left}}^{x_{\rm right}} {\rm d}x \, |\phi(x)|^2 = 1 \, . \tag{22}$$

Program 1: `http://www.physics.buffalo.edu/phy410-505/topic5/schroedinger.cpp`

```
void normalize() {
  double norm = 0;
  for (int i = 0; i < N; i++)
    norm += phi[i] * phi[i];
  norm /= N;
  norm = sqrt(norm);
  phi /= norm;
}
```

Here is the `main` function:

Program 1: http://www.physics.buffalo.edu/phy410-505/topic5/schroedinger.cpp

```
int main() {
  cout << " Eigenvalues of the Schroedinger equation\n"
       << " for the harmonic oscillator V(x) = 0.5 x^2\n"
       << " ----------------------------------------\n"
       << " Enter maximum energy E: ";
  double E_max;
  cin >> E_max;

  ofstream levels_file("levels.data");
  ofstream phi_file("phi.data");

  // draw the potential
  for (int i = 0; i <= N; i++) {
    double x = x_left + i * h;
    levels_file << x << '\t' << V(x) << '\n';
  }
  levels_file << '\n';

  // find the energy levels
  cout << "\n Level       Energy       Simple Steps   Secant Steps"
       << "\n -----   --------------   ------------   ------------\n";
  int level = 0;                  // level number
  double E_old = 0;               // previous energy eigenvalue
  E = 0.1;                        // guess an E below the ground state
  do {

    // estimate next E and dE
    double dE = 0.5 * (E - E_old);
    E_old = E;
    E += dE;
```

To find the root of $f(E)$ we will start by using the simple search root finder from the `cpl` library. This is the safest routine to use to locate the root approximately.

The key to success is to select a starting E that is below the desired root, and a step size dE that is smaller than the energy level spacing: then the simple search method is guaranteed to succeed.

Program 1: http://www.physics.buffalo.edu/phy410-505/topic5/schroedinger.cpp

```
    SimpleSearch simple;          // simple search object
    double acc = 0.01;            // set a relatively low accuracy
    simple.set_accuracy(acc);
    E = simple.find_root(F, E, dE);
```

Now that the root has been located approximately, switch to the much more efficient secant search method[2].

Program 1: http://www.physics.buffalo.edu/phy410-505/topic5/schroedinger.cpp

```
    SecantSearch secant;          // secant search object
    acc = 0.000001;               // can use a relatively high accuracy
    secant.set_accuracy(acc);
    double E1 = E + 100 * acc;    // guess second point required by secant
    E = secant.find_root(F, E, E1);
```

Output the results.

Program 1: http://www.physics.buffalo.edu/phy410-505/topic5/schroedinger.cpp

```
    cout.precision(10);
    cout << setw(4) << level++ << "  "
         << setw(15) << E << "  "
         << setw(10) << simple.get_steps() << "     "
         << setw(10) << secant.get_steps() << endl;
    levels_file << simple.find_root(q, x_left, h) << '\t' << E << '\n';
    levels_file << simple.find_root(q, x_right, -h) << '\t' << E << '\n';
    levels_file << '\n';
    normalize();
    for (int i = 0; i <= N; i++) {
      double x = x_left + i * h;
      phi_file << x << '\t' << phi[i] << '\n';
    }
    phi_file << '\n';
  } while (E < E_max);

  levels_file.close();
  phi_file.close();

  // output the search function to a file
  ofstream search_file("F.data");
  E = 0.1;
  double dE = 0.01;
  while (E < E_max) {
    search_file << E << '\t' << F(E) << '\n';
    E += dE;
  }
  search_file.close();

  cout << "\n Energy levels in file levels.data"
       << "\n Eigenfunctions in file phi.data"
       << "\n Search function in file F.data" << endl;
}
```

Homework Problem

Compare the numerical wavefunctions generated by the program `schroedinger.cpp` to the analytical formulas in your quantum mechanics textbook.

Add a term x^n to the harmonic oscillator Hamiltonian. Consider the cases $n = 3, 4$. Generate the spectrum of eigenvalues numerically and explain the changes you observe.

References

[1] B. Numerov, Publ. de l'Observ Astrophysique Central de Russie, **2**, 188 (1933).

[2] W.H. Press, S.A. Teukolsky, W.T. Vetterling and B.P. Flannery, "Numerical Recipes in C" (Cambridge University Press 1992), §9.2 Secant Method, False Position Method, and ridder's Method, `http://www.nrbook.com/a/bookcpdf/c9-2.pdf`.