Michael Eichberg: eichberg@itp.uni-frankfurt.de

Project part II

1 Overview

The goal for the second part of the programming project is to numerically solve the stationary 1-dimensional Schrödinger equation for the infinite potential well and to compute the corresponding quantized energy levels. Even though the infinite potential well can be solved analytically, it is useful to test your code for the final third part of the project, where you will consider a potential, for which the Schrödinger equation cannot be solved analytically.

2 Assignment

The stationary Schrödinger equation of a particle in 1 spatial dimension is given by

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

where m is the mass of the particle, $V(r)$ is the potential and E is an energy eigenvalue. Eq. (1) can be solved for any value for E. However, only solutions $\psi(x)|_{E=E_n} \equiv \psi_n(x)$, which fulfill specific boundary conditions, are normalizable and, thus, physically meaningful. These boundary conditions lead to quantized energy levels E_n. These energy levels E_n can be computed numerically, using the shooting method, as discussed in the lecture.

As already said in the overview, consider an infinite potential well of width L:

$$V(x) = \begin{cases} 0 & \text{for} \quad -L/2 \leq x \leq +L/2 \\ \infty & \text{otherwise.} \end{cases} \tag{2}$$

For $x < -L/2$ and $x > +L/2$, the wave function has to vanish. Since the wave function must be continuous, the boundary conditions are

$$\psi(\pm L/2) = 0\,. \tag{3}$$

In the lecture, dimensionless quantities $\hat{x} = x/L$ and $\hat{E} = \frac{2mL^2E}{\hbar^2}$ have been introduced. With these definitions the Schrödinger equation for the infinite potential well can be written as

$$\psi''(\hat{x}) = -\hat{E}\psi(\hat{x})\,. \tag{4}$$

It is restricted to the interval $-1/2 < \hat{x} < +1/2$, with boundary conditions

$$\psi(\hat{x} = \pm 1/2) = 0\,. \tag{5}$$

The eigenvalue equation (4) can be rewritten as a system of three first order ODEs, as discussed in the lecture.

(i) As a preparatory step, implement the Newton-Raphson algorithm discussed in the lecture to compute the root of a function $f(x)$. Compute the needed derivative using finite differences

$$\frac{\mathrm{d}}{\mathrm{d}x}f(x) \approx \frac{f(x+h) - f(x-h)}{2h}\,.$$

Test your implementation for $f(x) = \sin(x)$ with $h = 10^{-6}$ using double precision floating point numbers to numerically estimate the analytically known root $x_0 = \pi$. Start the root finding at $x_{1,a} = 3.0$ and $x_{1,b} = 3.5$, respectively, and count the number of iterations needed, to reach six digits of accuracy. Compare the number of Newton-Raphson steps to the 19 iterations required with the bisection method presented in the lecture (slides *Kontrollstrukturen*).

(ii) Write a function, which solves the Schrödinger equation using the 4th order Runge-Kutta method for given energy $\hat{E}$, with initial conditions $\psi(\hat{x} = -1/2) = 0.0$, $\psi'(\hat{x} = -1/2) = 1.0$ and step size $\tau = 10^{-3}$. Perform a scan of the violation of the boundary condition $\psi(\hat{x} = +1/2) = 0$ for energies $\hat{E} \in [1.0, 250.0]$ in steps of $\Delta\hat{E} = 1.0$, to find suitable starting energies for the shooting method (in other words, generate the analog of Figure 3 in the lecture notes with higher resolution). Read off from your plot crude estimates for the first five energy levels.

(iii) Use these estimates as starting points for the shooting method to compute the first five energy levels $\hat{E}_n$, $n = 1, 2, \ldots 5$, of the infinite potential well with at least six digits of accuracy. You can do that by comparing your results to the analytically known energy levels $\hat{E}_n = \pi^2 n^2$. Generate plots of the corresponding five wave functions similar to Figure 4 (bottom) in the lecture notes.

(iv) Now vary each of your numerical parameters, in particular the RK step size τ and the finite difference h, individually by at least factors of $1/2$ and 2, respectively, to confirm that your numerically obtained energy eigenvalues are stable and indeed agree with $\hat{E}_n = \pi^2 n^2$ in their first six digits. Then successively increase τ by the factor 2 until you find a deviation from the analytical exact result within the first six digits (provide this value of τ).

3 Code design

In the first part of the project, you implemented the 4th order Runge-Kutta algorithm. Reuse your files `rk.h` and `rk.c` (there is no need to change anything, except for the size `__N__` of the vector $\boldsymbol{y}(t)$).

Add two new files to your project, `newton_raphson.c` and `newton_raphson.h`, which contain the Newton-Raphson method, but nothing problem specific, i.e. related to quantum mechanics or the shooting method (the idea is that you can use these two files as a library in any other project). These files should contain the following variables and functions:

- ```
  const int max_steps = 1000;
  ```

  The value of this constant global variable corresponds to the maximum number of iterations, after which the Newton-Raphson algorithm aborts in case a root has still not been found.

- ```
  // Delta x = - f(x) * (2*h) / (f(x+h) - f(x-h))
  double newton_raphson(double x, double h, double epsilon, double (*
      function_ptr)(double));
  ```

  This function uses the Newton-Raphson algorithm to numerically compute the root of a function $f(x)$ provided by `function_ptr`. The algorithm starts at position x and uses numeric differentiation to determine $\Delta x = -f(x)/f'(x)$. The position of the root is returned, as soon as $\Delta x < \epsilon$, indicating convergence. If the algorithm does not converge within `max_steps`, a warning message should be printed to `stderr` and `NAN` (referring to *not a number*, defined in `math.h`) returned. One can check for `NAN` using the function `isnan()` also defined in `math.h`.

  Hint: To debug your program and monitor your numerical analyses, it is useful to provide extensive output to `stderr` (as e.g. done in the bisection function discussed in the lecture).

The `main` function of your program should be part of the file `infinite_well_shooting.c`. In that file, you also have to define a function `void compute_f_tau(double *f_tau, const Y_T *y_t, double tau)` specifically written for the infinite potential well, which can be passed to `step_rk4` in `rk.c` (for details, see the first part of the programming project). Moreover, define and implement the following variables and functions:

- `const double tau = 1.e-5;`

  A global variable to set the Runge-Kutta step size.

- `double compute_wave_function(double E, FILE *file);`

  This function computes the wave function for energy $\hat{E}$ and initial conditions $\psi(\hat{x} = -1/2) = 0$ and $\psi'(\hat{x} = -1/2) = 1$ using the function `step_rk4` defined in `rk.c`. It returns $\psi(\hat{x} = +1/2)$. If `file` is not `NULL`, the function prints the resulting wave function via `fprintf` using `file` as argument.

- `double bc_violation(double E);`

  This function computes the violation of the boundary condition at $\hat{x} = +1/2$ for given energy $\hat{E}$. This can be done in a simple way by just calling the function `compute_wave_function`. Note, that `bc_violation` has the signature `double (*)(double)` and can be passed to the function `newton_raphson` defined in `newton_raphson.c`. At the end, the violation of the boundary condition at $\hat{x} = +1/2$, i.e. $\psi(\hat{x} = +1/2)$, is returned.

- `void energy_scan(double E_min, double E_max, double delta_E);`

  This function performs a scan of the violation of the boundary condition at $\hat{x} = +1/2$ in the energy range $\hat{E}_{\text{min}} \leq \hat{E} \leq \hat{E}_{\text{max}}$ in steps of $\Delta\hat{E}$. It calls the function `bc_violation` for each energy value and prints the results to `stdout` (such that they can easily be redirected to a file using `>` in the terminal).

- `double compute_energy_level(double E_start, double h, double epsilon);`

  This function computes an energy eigenvalue using the Runge-Kutta shooting method. `h` and `epsilon` are parameters used in the Newton-Raphson root finding. The function returns the found energy eigenvalue, or `NAN` in case the root finding does not converge.