Physics 3340/8300-005 - "Computational Physics" - Fall 2013
Final Project
Assigned: Friday, 6 December
Due: Friday, 13 December 11:59pm by Blackboard upload

Reading:

Cheney and Kincaid sections 3.1, 7.1, 7.2, 7.4, 11.1 (6th ed.: 3.1,10.1, 10.2, 11.1, 11.2, 14.1)
http://www.physics.smu.edu/fattarus/NumericalCTutorial.html
http://www.physics.smu.edu/fattarus/RootsLab.html
http://www.physics.smu.edu/fattarus/DiffeqLab.html

Problem:
Find the allowed electron energy bands and forbidden zones for a discrete one-dimensional approximation to a crystal. A description of a customized one-dimensional periodic electron potential for your individual project will be emailed to you. The specification of the potential profile will consist of the number of periodic wells, the (negative) potential electron energy at the bottom of each well, the width of the well, and the periodic spacing between the well centers. Follow the lecture slides for the numerical and coding details you will need, and submit a final report detailing your work by Blackboard upload. The project should be separated into three steps toward the final solution:

1) Produce a gnuplot of your periodic electron potential. First write a C function `potential()` that calculates the potential as a function of spacial coordinate *x* according to your custom specification. Then write a main program similar to that in the numerical C lab at the link given above that links to the sweep.c class tool routine and calls the `sweep()` function to produce a data file describing the potential as a function of *x*. Use a fine enough *x* increment value to outline all the corners of the potential. Your C code and gnuplot image should be included in your final project report. Hint: Consider the usefulness of the C double precision expression

```
x - period * round(x/period)
```

in evaluating a function that is periodic in *x*.

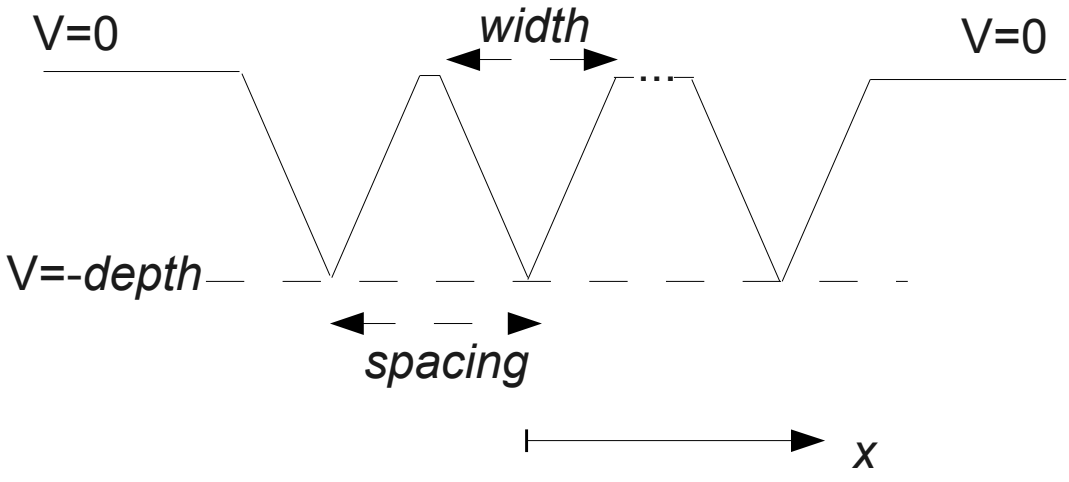
2) Produce a gnuplot of the wave function error function, as discussed in class. The roots of this function will be the electron energy eigenvalues. Write a C function `wave_shoot()` that accepts an argument of a trial electron energy, calculates the corresponding wave number *k*, initializes the wave function and its derivative at the minimum value of *x* and calls the `rk4_quiet()` function to integrate the time-independent Shroedinger equation in one dimension through the periodic potential with increasing *x* values. (Recall that the `rk4_quiet()` function is basically the same as the `rk4()` function we have been using in class, but does not feed the differential equation solution progress to the standard output stream. It does, however, leave the final state variable values in the `y[]` array.) As described in the lecture slides, the final wave function value at the end of the potential periods is the return value of this function, and represents the "error" with respect to the zero value that will give an approximately normalizable wave function, which is a requirement for any Schroedinger equation solution to be physically meaningful. This `wave_shoot()` function will be similar to parts of the main program code used in the differential equation lab at the link given above. Your C function `wave_shoot()` should call the `rk4_quiet()` function with a pointer to another function `schroedinger()` that assigns values to the state variable derivatives to solve the Schroedinger differential equation. (That `schroedinger()` function will also make use of the function `potential()` you wrote in part 1 that calculates the electron potential as a function of *x*.) Use a fine *x* step value, such as 1/100th of the width of *each potential well*, to allow the R-K4 algorithm to accurately trace the differential equation

solution through the array of wells. The main program for this part should again call the `sweep()` routine to produce a data file describing the error value as a function of electron energy. Use a rather fine increment value for the trial electron energy, such as 1.0e-4 eV, to capture all of the fine oscillations around zero. Assume a mass (actually mc^2) of the electron of 0.5110 MeV. Don't be surprised if this program takes a few minutes of execution time on most computer hardware. Then produce a gnuplot of the error function zoomed into the wave function value range of -2 to +2 as in the class slides. You should be able to see the roots of this wave error function clustered in allowed energy bands. Include the C code and gnuplot image in the final report.

3) Now use the `sweep_regula_falsi()` routine to find the electron energy eigenvalues for your crystal, which you should see segregated into allowed energy bands and forbidden zones. Write another main program that uses the same structure as the main program in part 2, but instead of passing a pointer to the wave error function `wave_shoot()` as an argument to `sweep()`, passes it instead to `sweep_regula_falsi()`. This should find all the roots of the function you plotted in part 2 and send a list of these allowed energy levels to the standard output stream. Refer back to the root finding lab at the link given above for example code that calls functions in the sweep_roots.c package. To make sure you find all the roots, the sweep increment you pass to `sweep_regula_falsi()` must be fine enough, such as 1.0e-5 eV. A root convergence tolerance of 1.0e-9 eV and a maximum iteration count of 500 are good values to use. Note that this program will probably take at least several minutes of execution time on most computer hardware. Include the C code for this part and a copy of the text output with all the eigenvalues in your final report.

Hints:

You should not only write your `potential()` function in part 1 to calculate the pattern of potential wells and any spacing between them at 0 eV as a function of x, but you also need to write it so that for x argument values outside the periodic pattern of wells in your crystal, to the left and to the right on the x axis, the potential value the function returns is flat and up at 0 eV. This is necessary to provide a region of flat potential at both ends where we know the wave function solution of the Schroedinger equation will follow the linear combination of exponentials. When the solution of $\psi(x)$ follows the exponentials toward the extremes of the x range, as shown in the lecture slides, it will be possible to find a normalizable solution.



The `wave_shoot()` function should integrate the Schroedinger equation from an *x* value of *width* before the center of the first well (or in other words, one half *width* before the first well corner) to an *x* value of *width* after the center of the last well in the crystal, where *width* is the width of your potential wells. This will be consistent with the lecture slides that detail the initial values of the $\psi(x)$ and its first derivative.

Your `schroedinger()` function will need to know the value of electron energy that the Schroedinger equation is being solved with. The argument list of this `schroedinger()` function is fixed by the function type that the `rk4_quiet()` function has been written to call. So an easy way to provide the value of electron energy to the `schroedinger()` function is to define a global variable at the top of the program file above where any function is defined to hold the current trial energy value. You may assign the current energy value to this global variable before the `schroedinger()` function is called and it will be available inside the `schroedinger()` function. This is essentially the method used in the lab exercise programs to assign values given on the command line that may be referred to inside functions. For example, see the ballistic.c program in the differential equations lab to see how the values of the projectile mass, the initial launch speed and the launch angle are collected from the command line, assigned to global variables defined at the top of the source code file, and used inside the `ballistic()` function, which is the counterpart to the `schroedinger()` function in this final project.