Session_05 activities worksheet

Online handouts: eigen_tridiagonal.cpp printout, eigen_basis.cpp printout, harmonic_oscillator.cpp printout, nan_test.cpp printout, and square_well.nb Mathematica notebook.

Your goals for these activities:

1. Take a first look at your shell start-up file.
2. See some examples of nan's and inf's.
3. Use the eigen_tridiagonal program to look at eigenvalues of the harmonic oscillator potential.
4. Find the lowest bound-state eigenvalues of two familiar potentials using the eigen_basis program.
5. Examine (and try to understand) how the accuracy of your results depends on the size of the harmonic oscillator basis and the choice of the basis parameter b.

A) Optional: Aliases in Your Start-up Shell

The shell is the program you type to at the command line. The two most popular ones are bash and tcsh. You can figure out which of the two you are using by typing `echo $0`. There are many good things to learn about the shells. Today we'll just learn about aliases. Just follow the corresponding instructions for bash (tcsh).

1. **Bash (tcsh) aliases.** The .bashrc (or .cshrc.more) file sits in your home directory and is "sourced" when you start an interactive terminal. Take a look at the sample .bashrc (or .cshrc.more) file printout. These files, without the ".", are in the Session_05 zip file. Copy any aliases of interest to your own .bashrc (or .cshrc.more) or the whole thing (before you copy the whole thing make a backup of the old one via `cp ~/.bashrc ~/.bashrc.orig` or `cp ~/.cshrc.more ~/.cshrc.more.orig`) using `cp bashrc ~/.bashrc` (or `cp cshrc.more ~/.cshrc.more)` if you don't have one already (WARNING: do not use these cp commands if you alrady have a .bashrc or .cshrc.more file as they likely overwrite what you alreay have!) Activate the changes with the command: `source ~/.bashrc` (or `source ~/.cshrc`). *Try some aliases such as* ll *(for "long listing") and* df, *which shows info about the disk file system. Do they work? Questions?*

   → Yes, it works, and I don't have any questions.

2. *Create your own alias.* E.g., you could make an alias to change from your start-up (login) directory to your 5810 working directory. Be creative! *What line did you add to the start-up file?*

   → I added a command to automatically CD the directory to my. phy5810 directory. I also added something that auto-starts gnuplot in the background with gnuplot &.

B) Nan's and Inf's

Just a quickie: Take a look at nan_test.cpp, use make_nan_test to create nan_test and then run it.

- *What do you think the result of 0.*(1./0.) will be? Predict and then modify the code to check it out. [Note: Use the variables "numerator" and "denominator" to calculate (1./0.).]*

→ I predict that the value of 0.*(1./0.) will be nan, since 1./0. will be inf, and 0.*inf is indeterminate, thus giving us nan. When modifying the code, we do indeed get nan.

C) Bound States by Matrix Diagonalization in Coordinate Representation

The program eigen_tridiagonal.cpp uses the GSL library routines explored in eigen_test.cpp (Session _04) to find the eigenvalues and lowest eigenvector of the harmonic oscillator using a method described in the Session_05 notes. With the units here, the lowest eigenvalue should be (3/2)hbar-omega = 1.5 (read comments in code!).

1. Using make_eigen_tridiagonal, compile and run the code a few times to see how it works. Try various values of Rmax and N such as Rmax=3, N=20 or 50 (*make a chart here of Rmax, N and the two lowest eigenvalues for five pairs of Rmax,N*). *How can you verify that the code is working?*

Rmax	N	Lowest Eigenvalue	2nd Lowest Eigenvalue
3	20	1.681778	5.083030
3	50	1.684067	5.113673
5	20	1.495431	3.494389
5	50	1.499542	3.515748
5	70	1.499924	3.517728

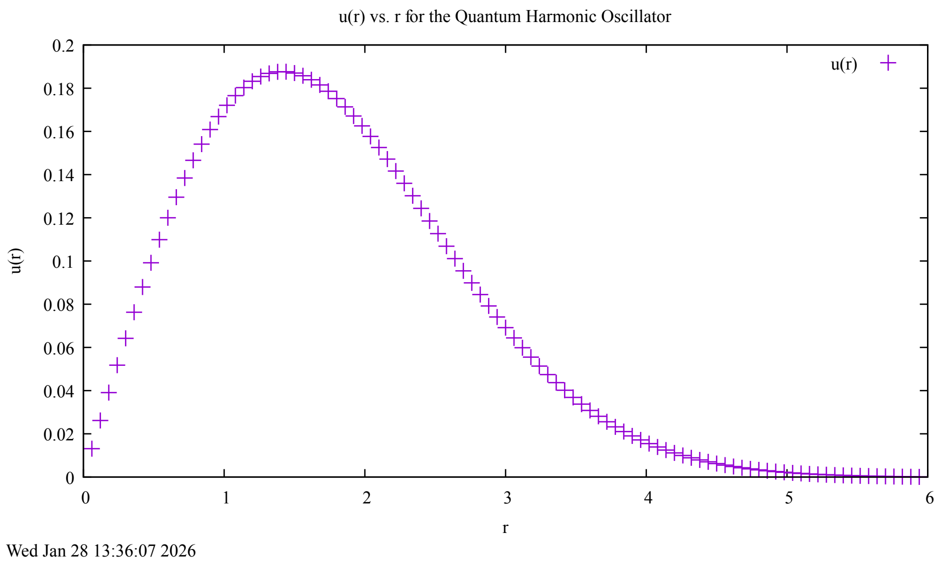
→ To verify that this code is working, the best way is to see the convergence of the lowest eigenvalue. For a fixed Rmax, as we increase the steps N, we should get that the lowest eigenvalue converges to our theoretical prediction of 1.5 (given the units) and the 2nd lowest should converge to E1=2n+1.5 which would be 3.5. Seeing the Rmax=5 parts of the table, we do verify this convergence, which is a good sign that our code is working. We also note that we are getting the same number of eigenvalues as expected (N-1). Another way to test this is working is to output more than two eigenvalues and ensure they are evenly spaced, as theorized by the harmonic oscillator.

2. Change the code so that only the lowest few eigenvalues are printed out. Look at the output file eigen_tridiagonal.dat and plot it with gnuplot. *What is the physical meaning of this function? The value at r=0 is not given; what should it be?*

   → The physical function is the reduced radial equation of the harmonic oscillator (the radial part of the separation of variables of the wave equation). The value at r=0 by this should be u(r)=0.

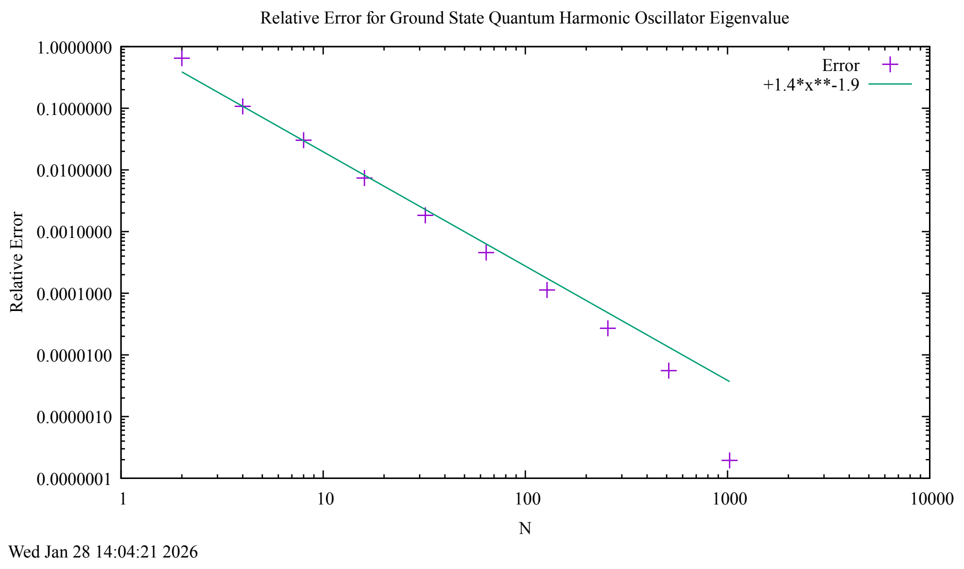
3. You need to pick a reasonable value of Rmax. *Justify your choice based on the eigen_tridiagonal.dat plot:*

   → A reasonable Rmax (looking at the .dat as well) would be 6. I chose this because the peak of the graph will be around 1.5 (eigenvalue), so to see the entire graph we will want to capture the entire graph. So, a reasonable value would be around 3 times this value, so I chose Rmax=6 with an N of 100. Below is the graph:



4. For your choice of Rmax, try N = 4,8,16,32,64,128,256,512,1024 (you could add a loop to calculate these). *How does the relative error for the lowest eigenvalue scale with N?* Attach an appropriate plot to validate your answer.

   → The relative error scales as a power law as a function of N. It appears that the rough power law is that the error scales to the power of -2 as a function of N. Below is the plot:

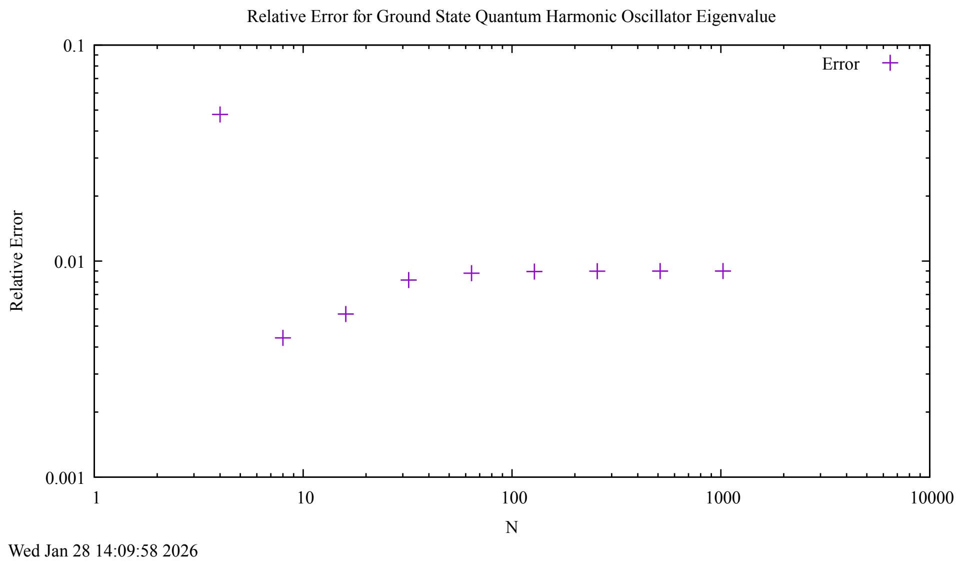


5. *Explain the slope you found based on the approximation to the second derivative.*

→ The slope we got was -2, or a power law of 1/x^2. This does match the approximation of the second derivative, where d^2u/dr^2 is proportional to h^2, and h^2 is proportional to N^-2. Hence, the second derivative, or this slope, should be proportional to N^-2, which it is.

6. *Bonus: repeat with Rmax = 4 and explain what happens to your graph.*

→ Repeating for Rmax = 4 gives us a graph that decreases in error before plateauing. This makes sense because at first the error decreases by the power law. However, since our result is cut off by a bad Rmax value that is too small, the error flattens, because the output is truncated earlier. The Hamiltonian we are solving is incomplete, so the error will not decrease further, even if N is extremely large. Below is the plot:



D) Bound States from Diagonalizing the Hamiltonian in a Basis

The program in eigen_basis.cpp uses the GSL library routines to diagonalize (i.e., to find the eigenvalues and eigenvectors) a Hamiltonian matrix in a basis of harmonic oscillator wave functions. You may want to refer to the GSL handout on eigensystems (there is also a printout of eigen_basis.cpp).

The eigen_basis program uses units with the particle mass=1 and hbar=1. The program asks you to choose

- a potential (Coulomb or square well);
- the parameter b for the harmonic oscillator basis (see harmonic_oscillator.cpp for the definition);
- the number of basis states to use.

The parameters of the potentials are fixed in the code. The eigenvalues for the Hamiltonian matrix are written to the terminal sorted in numerical order (as opposed to absolute-value sorting, which was used in eigen_test.cpp). The corresponding eigenvectors are generated but are not printed out (that is, the print statements are commented out).

The Coulomb potential is defined with Ze^2=1, which means that the Bohr radius is also unity. This means that the exact bound energy levels are given by $E_n = -1/2n^2$, with n=1,2,...
The square well potential is defined with radius R=1 and depth $V_0 = 50$. *You should find that there are three bound states.*

Here are some subgoals. There won't be time for everything (as usual) but you'll have a chance to finish them as part of a future Homework.

1. Run the Mathematica notebook square_well.nb directly on the laptops (i.e., not on the virtual machine). Make sure you understand what it is doing; e.g, look up FindRoot in the Help Browser. *Find the bound-state energies for the square well parameters used here (you need to change the notebook parameters!).*

   → Using Mathematica for our system, we find the three bound-state energies. We get the following: E = -45.9321, -33.8733, and -14.5248.

2. Compile and link the code eigen_basis using make_eigen_basis. This also compiles harmonic_oscillator.cpp. Run it a few times with each of the potentials to get familiar with it. If you try too large a basis size, the run time may be too long (so start small!). Look through the printout to see the basic idea of how the code works and find where the equation for the matrix element is implemented.

3. *Based on the "exact" results from Mathematica, which of the approximate eigenvalue(s) for the square well are most reliable? Why do you think this is?*

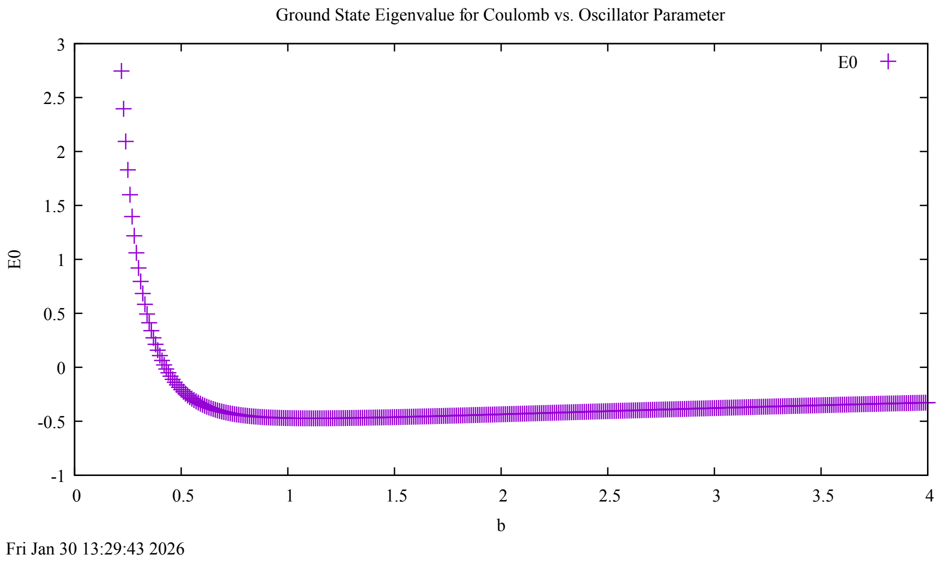
→ I believe the most reliable square well eigenvalue is the lowest one, which in this case would be -45.9. This is because this is the ground-state eigenvalue, which is always the most stable and well-converged, whereas the higher-energy states require more oscillations to describe accurately. Also, the ground state wave function is concentrated at the bottom of the well, keeping it more reliable.

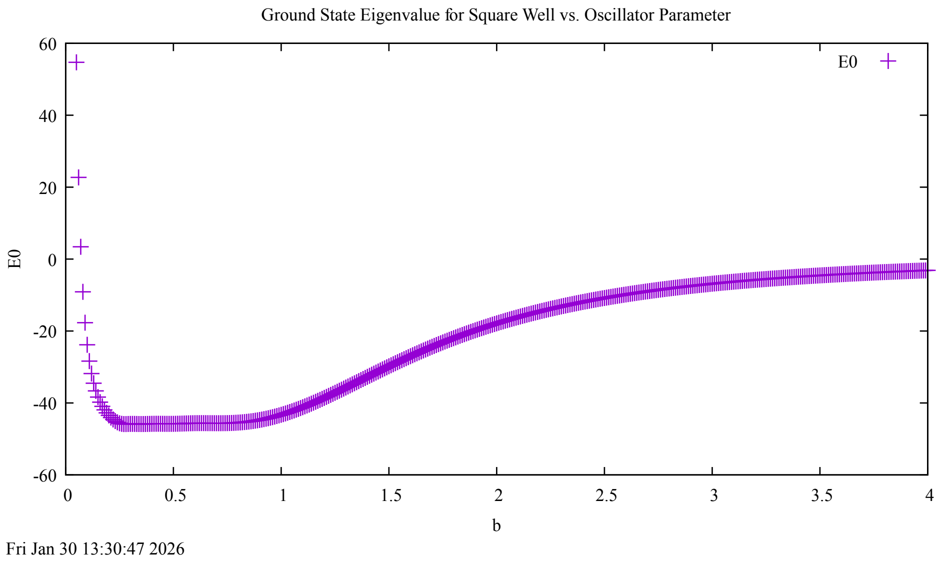
4. *Considering all three of the lowest eigenvalues, which are calculated most effectively, those of the Coulomb potential or the square well potential? Can you explain your observation?*

   → The eigenvalues calculated the most effectively are most likely the square well potential. This is because the ground state for the square well is a smooth sine-curve even though the potential itself has a jump. The coulomb potential has a discontinuity at r=0 with a sharp cusp, which would make it mathematically difficult. We can see this by increasing the number of dimensions for some fixed b, and seeing that the lowest eigenvalues converge faster for the square well.

5. You have under your control the size of the basis (i.e., the dimension of the matrix) and the harmonic oscillator parameter b (see harmonic_oscillator.cpp for the definition). For a fixed basis size (pick one that reproduces the ground state reasonably), how do you find the optimum b? (Hint: think gnuplot!) *Can you qualitatively (or semi-quantitatively) account for your result? (Think about the potentials and guess what the lowest wave functions will look like and what changes about the basis when the harmonic oscillator parameter b is changed.)*

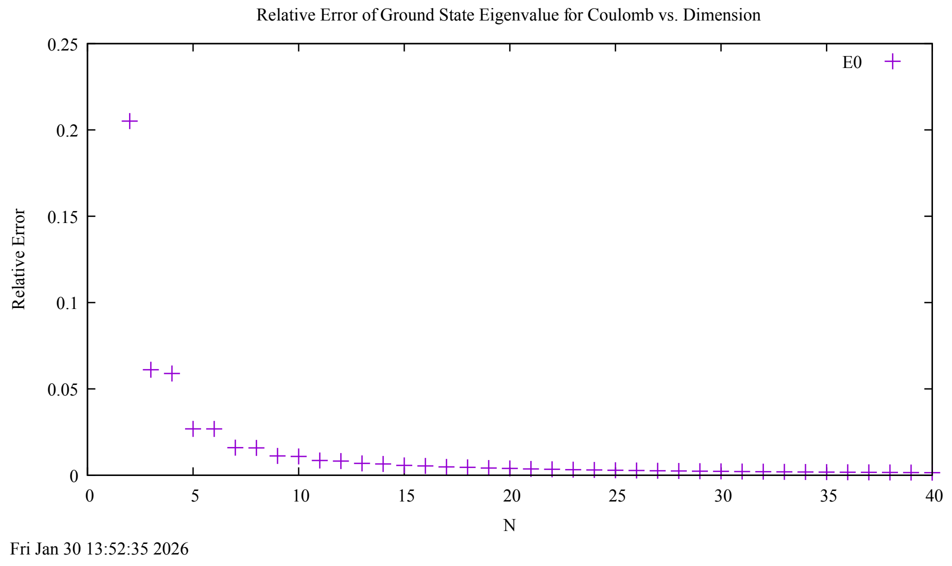
   → We can find the optimal b by iterating b values and then calculate E0, the ground state energy eigenvalue. When the slope of the returning graph is 0, that would be the optimal b value, picking the smallest part. We can also account for our result by looking at our potentials. Given our choice of units, we would anticipate for the Coulomb potential a b of about 1.0, and for the square well we would anticipate a value roughly around there as well, given the formulas in the notes. Making the plots, we can see for the Coulomb potential, we do get an optimal b of about 1.0. For the square well, we get a value closer to 0.5, but that can likely be accounted for as that b value being better at containing the wave in the square well, making the Gaussian less "fat". Below are the two plots:

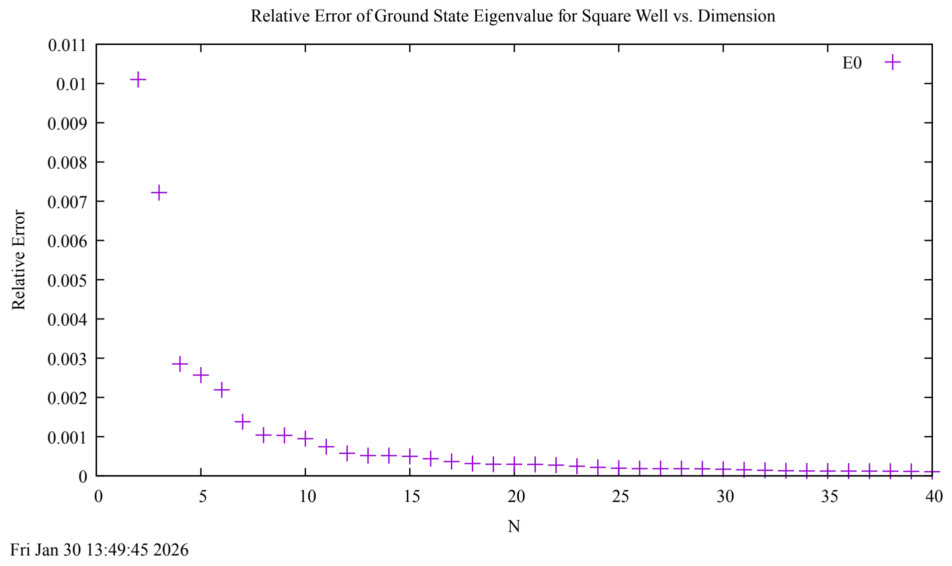




6. *If you now fix b (if you have time you can consider two or three different values in turn), how can you find how the accuracy of the ground state energy scales with the basis size?* Make an appropriate plot.

→ We can iterate the values of N dimension size for a fixed b, like we did for the previous question. We can calculate roughly what the exact energy for the ground state should be for our given choice of constants and units. In doing so, we can find the relative error of the ground state energy calculated versus the exact value. Doing that yields the following plots, which shows us that we get pretty good precision in a low number of dimensions for the square well, while the Coulomb potential requires some more dimensions. This also backs up our response for 4.





7. Look at the code. *How could you make it more efficient? (What do you think is the limiting factor based on the scaling of the time with the size of the basis?) For example, could you speed it up by almost a factor of two?* (Hint, hint!)

   → You could make the code more efficient by utilizing the symmetry of the Hamiltonian matrix. In QM, the Hamiltonian operator is Hermitian, meaning the matrix is symmetric. The nested loop used to calculate the matrix elements individually could be simplified to only calculate the upper or lower triangle of the matrix. This would reduce the number of iterations by roughly half, leading to a speedup factor of approximately 2.

8. *(For Homework_03) How would you find the wave function that corresponds to a given state (e.g., the ground state) (hint: it involves the eigenvector)?* You can only describe your approach here and do not need to add code yet, but in Homework_03 you need to code it up.

→ To find the wave functions that corresponds to a given state, you would want to first select the eigenvector of interest. Then, each element in the eigenvector corresponds to some basis function, and then you would construct a linear combinations of the eigenvector elements times the radial basis functions. Then to evaluate, for every r on the grid you would define a grid and sum.

E) Reflection

Write down which of the six learning goals on the syllabus this worksheet addressed for you and how:

→ This session helped with many of the learning goals. Alongside the typical learning to write clear, documented code and understanding tools of computational physics through GSL, I learned a lot about computational solutions to physics problems. I also am starting to have a better understanding of physics by having to put physics concepts into code, helping me refresh my quantum mechanics knowledge.