Software Spec –BoundRSESolver Class

This class solves the radial Schrodinger equation to obtain the wave function Pnl(r) and energy eigenvalue E for bound states with (n,l) for a given potential function v(r) tabulated on a IRadialMesh.

This class supports asynchronous operation on multiple threads.

# Background

The differential equation to solve is:

|  |  |  |
| --- | --- | --- |
|  |  |  |

For Enl < 0 (bound state).

Numerov’s method is useful for solving DEs of the form:

|  |  |  |
| --- | --- | --- |
|  |  |  |

When expressed in this form, the Numerov function *v*(*r*) is:

|  |  |  |
| --- | --- | --- |
|  |  |  |

Note that since V(r) is only a function of the total radial charge densities summed over n and l, it does not need to be modified as E changes. However, since the solution is computed for an electron of (E,n,l) moving in the average field of all other electrons, the V(r) does depend on the particular orbital being solved for. E can be changed independently, which is very convenient for the energy eigenvalue solver.

Since the return value of the algorithm is a structure (needs to return E, Pnl, and a flag indicating success or failure), this is best implemented as a class that can be configured and then invoked with a Solve() method on a worker thread.

Inputs:

Requires the mesh as IRadialMesh, the potential V(r) to solve with, the primary quantum number N (used to count zero crossings), the angular momentum quantum number L, the occupancy (used only to construct the Orbital return value) and tolerance criteria for the solution.

Outputs:

An Orbital instance containing N, L, E, P(r), and occupancy, and a flag indicating success or failure of the routine to converge on a solution.

Implementation:

The number of zero crossings that the wave function should have is

|  |  |  |
| --- | --- | --- |
|  |  |  |

If the outward integration reaches the point where E – V(r) < 0 (Rcross) and Ncross is too small or too large, we need to modify the bracketing values of E and iterate. It would be good to add in a check to make sure the value of the wave function hasn’t exceeded a very large number before reaching Rcross.

The routine searches for the solution using the following algorithm:

1. Construct the solver potential from V(r) and L
2. Calculate Rcross (where V(r) = E)
3. Bracket the energy range for the solution
   1. Start with Emin = Vmin, Emax = 0. Won’t actually have to solve at those limits, we know those don’t work
   2. First trial will be Vmin/2
   3. If zero crossings are too large, set Emin = Vmin / 2, Emax = 0, Etrial = Vmin / 4
   4. If zero crossings too few, set Emin = Vmin, Emax = Vmin/2, Etrial = 3Vmin/4
4. Perform a binary search until the proper number of zero crossings is achieved and solution does not zoom out to an infinity before reaching Rcross.
   1. Integrate outward with Etrial
   2. It is not necessary to compute integral or logarithmic derivatives
   3. Count zero crossings
   4. If the number of zero crossings is too small, increase Etrial. If too large, decrease.
   5. Continue until the number of zero crossings is proper.
5. Once zero-crossings are correct, switch to inward/outward matching procedure
   1. Requires both outward and inward integration, and computation of integrals and logarithmic derivatives of both.
   2. Can still use bracketing strategy of binary search, but criteria is matching and the distance to move the brackets is no longer binary, but is determined by delta E calculation (or perhaps binary, whatever is easiest).

Next Steps:

* Need a function to compute Rcross for a given V(r), E.

Public Class BoundRSESolver

Public Shared Sub Solve(mesh As IRadialMesh, N As Integer, L As Integer, occupancy As Double, V As Double()) As Orbital

Public Shared Sub Solve(mesh As IRadialMesh, N As Integer, L As Integer, occupancy As Double, V As Double(), tolerance As Double) As Orbital

I tend to prefer the shared method implementation for the implied thread safety and the ease of invocation.

The class should also expose a Delegate:

Public Delegate SolveDelegate(mesh As IRadialMesh, N As Integer, L As Integer, occupancy As Double, V As Double()) As Orbital

Using the method

Asynchronously invoking the Solve() method:

Dim result As IAsyncResult = SolveDelegate.BeginInvoke(mesh, N, L, occupancy, V, New AsyncCallback(SolveCallback), Nothing)

Private Sub SolveCallback(result As IAsyncResult)

* Get the delegate and call EndInvoke
* Take the return value and add it to the Orbitals list (is adding to the list a thread safe action?)
* Adding to list – get a SyncLock first!
* lock (collection.SyncRoot)  
  {  
     DoSomething(collection);  
  }