Software Spec – RadialSchrodingerSolver Class

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

This class should support asynchronous operation on multiple threads.

# Background

The radial Schrodinger equation is (in Rydberg units)

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

With the additional boundary condition that .

Where the *Pnl*(*r*) are the *r*-multiplied radial wave functions

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

Which are normalized such that

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

In the central field approximation (following Herman and Skillman),

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

Where the radial charge density is

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

Where *Nnl* is the occupation number for the orbital (n,l) with both spins, and the spherically averaged radial charge density is

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

## Latter Tail Correction

The correct potential to use in Eq 1 is slightly modified from that given in Eq 4:

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

Where Z is the atomic number and N is the number of electrons. The switching radius *r*0 is chosen such that

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

## V(r) function for Numerov

The differential equation to solve is:

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

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.

For a given n and l, we can define

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

where

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

Public Class Potential

This class computes the potential function v0nl(r).

Inputs – the collection of Pnl(r) functions for all orbitals, their occupancies (configuration object), the atomic number Z, and the radial mesh.

Alternate constructor – omits the Pnl(r) functions, signaling that an initial potential is desired.

Outputs – a collection of v0(r) for each (n,l) orbital in the configuration (function to return a copy of the array for thread safety and performance).

? Does the alpha term in the exchange potential depend on Z?

Requires: A routine to integrate a given radial function on the mesh.

Would be nice to implement IComparable or a difference operator so that can easily compute the convergence figure of merit between two instances:

isConverged = (newpotential – oldpotential) < tolerance.

Implements IComparable

Sub New(

Sub New(Z as integer, config as ElectronicConfiguration)

Sub New(Z as Integer, config as ElectronicConfiguration

Public Property WaveFunction(n,l) as Double()

Public Property Energy(n,l) as Double()

Public Function Sigma(n,l) as Double()

Public Property Configuration()

Public Sub ComputePotential()

Readonly Property Potential(n,l) as Double()

Readonly Property Mesh() as IRadialMesh

Atom has Orbitals corresponding to a Configuration. Atom has a Potential that is a consequence of the Orbitals.

OrbitalCollection <>-- Orbital, keyed by N\_L string

Wrap a collection, expose methods to retrieve orbital based on N, L

Orbital (Returned from RadialSESolver class)  
 P as Double()  
 N as Integer  
 L as Integer  
 E as Double  
 Occupancy as Double

Program Flow:

Create OldPotential (New Potential() or Potential(OldOrbitals))

New RadialSESolver(N, L, V(r), IRadialMesh, Etrial) {One solver for each orbital}

RadialSESolver.Solve() (thread safe) {Spin off thread for each orbital}

NewOrbitals.Add(RadialSESolver.GetOrbital) {Need Add method to be thread safe, lock if necessary}

When all NewOrbitals are calculated, create NewPotential = New Potential(NewOrbitals)

Compare to OldPotential and test for convergence

If Not Converged then

OldPotential = NewPotential

return to (1)

1. Create and test a RadialIntegrator class, based on trapezoidal rule
2. Create and test the Orbital class
3. Create and test the Potential class
4. Create and test the RadialSESolver class
5. Encapsulate the program flow above in Atom

Implements IRadialSchrodingerSolver

Interface for different methods of solving the r-multiplied radial Schrödinger equation. Given an input potential, n and l, solves for the energy eigenvalue and radial wavefunction. Supports asynchronous operation on multiple threads.

Solve(mesh as IRadialMesh, n as integer, l as integer, v as double()) as RadialSolution

Solve(mesh as IRadialMesh, l as integer, E as double, v as double()) as RadialSolution

Implementing this as a thread-aware class requires separating the inputs and outputs and making the function call a parameterless sub:

Sub New()

Sub New(mesh as IRadialMesh, n as integer, l as integer, v as double())

Sub New (mesh as IRadialMesh, l as integer, E as double, v as double())

Property mesh as IRadialMesh

Property n as Integer

Property l as Integer

Property v as Double()

Property u as Double()

Property E as Double

Property BoundState as Boolean

Readonly Property Result as RadialSolution

Readonly Property Finished as Boolean

Readonly Property Running as Boolean

Sub Solve()

Event SolverFinished()

Event SolverFailed()

Sub New()

Sub New(mesh as IRadialMesh, v as double(), u as double(), direction as IntegrationDirectionEnum, y0 as double, yprime0 as double)

Sub New(mesh as IRadialMesh, v as double(), u as double(), direction as IntegrationDirectionEnum, y0 as double, yprime0 as double, iend as integer)

Sub New(mesh as IRadialMesh, v as double(), direction as IntegrationDirectionEnum, y0 as double, yprime0 as double)

Sub New(mesh as IRadialMesh, v as double(), direction as IntegrationDirectionEnum, y0 as double, yprime0 as double, iend as integer)

Property mesh as IRadialMesh

Property v as Double()

Property u as Double()

Property y0 as Double

Property yprime0 as Double

Readonly Property y as Double()

Readonly Property yprime as Double()

Readonly Property Finished as Boolean

Readonly Property Running as Boolean

Sub Solve()

Event SolverFinished()

Event SolverFailed()

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

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

In our case,

With the additional boundary conditions that

{need another boundary condition here to ensure unique solution to within a constant}

Class NumerovIntegrator

Public Shared Function Integrate(mesh as IRadialMesh, v as double(), u as double(), direction as IntegrationDirectionEnum, y0 as double, yprime0 as double) as NumerovIntegrationResult

Public Shared Function Integrate(mesh as IRadialMesh, v as double(), u as double(), direction as IntegrationDirectionEnum, y0 as double, yprime0 as double, iend as integer) as NumerovIntegrationResult

Public Shared Function Integrate(mesh as IRadialMesh, v as double(), direction as IntegrationDirectionEnum, y0 as double, yprime0 as double) as NumerovIntegrationResult

Public Shared Function Integrate(mesh as IRadialMesh, v as double(), direction as IntegrationDirectionEnum, y0 as double, yprime0 as double, iend as integer) as NumerovIntegrationResult

Structure NumerovIntegrationResult

Sub New(y as Double(), yPrime as Double(), istart as integer, iend as integer)

Public Readonly Property y as Double()

Public Readonly Property yPrime as Double()

Public Readonly Property iStart as Integer

Public Readonly Property iEnd as Integer

Program flow

1. Potential V(r) is constructed for n, l
2. Search for solution
   1. Bracket the energy range for the solution
   2. Perform a binary search until the proper number of zero crossings is achieved
      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.
   3. 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.

Next Steps:

* Need a function to count zero crossings of an array of values from istart to iend
* Need a function to integrate an array y tabulated on IMesh
* Need a function to compute Rmatch and Imatch for a given V(r), E.