

Apr 10, 08 8:17

eigen_tridiagonal.cpp

Page 1/3

```

// file: eigen_tridiagonal.cpp
//
// Program to find bound state eigenvalues for a potential
// by discretizing and then diagonalizing the Hamiltonian
// in coordinate representation using the GSL eigenvalue/eigenvector
// routines.
//
// Programmer: Dick Furnstahl furnstahl.1@osu.edu
//
// Revision history:
// 01/23/07 original version
//
// Notes:
// * We follow the Session 5 notes for method 2 of solving the
// Schrodinger equation by matrix diagonalization in
// coordinate representation. This uses a finite difference
// approximation to the 2nd derivative.
// * We apply the method to a three-dimensional harmonic oscillator
// with angular momentum l=0.
// * We use units in which hbar = 1, k = 1/2, and M = 1/2.
// The eigenvalues for the 3d harmonic oscillator are then (l=0)
// hbar omega (2n + 3/2) = 2n + 3/2 for n = 0,1,2,3,...
// * Based on the documentation for the GSL library under
// "Eigensystems" and on Chap. 15 of "Computational Physics"
// by Landau and Paez.
// * Uses the GSL functions for computing eigenvalues
// and eigenvectors of matrices. The steps for this part are:
// * define and allocate space for matrices and vectors we need
// * load the matrix to be diagonalized (pointed to by Hmat_ptr)
// * find the eigenvalues and eigenvectors with gsl_eigen_symmv
// * sort the results numerically
// * print out the results
// * As a convention (advocated in "Practical C++"), we'll append
// "_ptr" to all pointers.
// * When sorting eigenvalues,
//   GSL_EIGEN_SORT_VAL_ASC => ascending order in numerical value
//   GSL_EIGEN_SORT_VAL_DESC => descending order in numerical value
//   GSL_EIGEN_SORT_ABS_ASC => ascending order in magnitude
//   GSL_EIGEN_SORT_ABS_DESC => descending order in magnitude
//
// To do:
// * Try other potentials.
//
// *****
//
// include files
#include <iostream>           // note that .h is omitted
#include <iomanip>            // note that .h is omitted
#include <fstream>
#include <cmath>
using namespace std;

#include <gsl/gsl_eigen.h>    // include the appropriate GSL header file

inline double sqr(double x) {return x*x;};

double V_ho(double r);

// ***** main program *****
int
main ()
{
    // Choose Rmax (maximum radius)
    double Rmax = 5.;
    cout << "Enter maximum radius (Rmax): ";
    cin >> Rmax;

    // Pick the value of the number of steps N

```

Apr 10, 08 8:17

eigen_tridiagonal.cpp

Page 2/3

```

int N = 0;
cout << "Enter the number of steps (N): ";
cin >> N;
// The matrix dimension is N-1 (see the notes).
int dimension = N-1;

// Calculate h = Delta x
double h = Rmax/double(N);
double hsq = sqr(h);

// See the GSL documentation for matrix, vector structures
// Define and allocate space for the vectors, matrices, and workspace
// original gsl matrix with Hamiltonian
gsl_matrix *Hmat_ptr = gsl_matrix_alloc (dimension, dimension);
// gsl vector with eigenvalues
gsl_vector *Eigval_ptr = gsl_vector_alloc (dimension);
// gsl matrix with eigenvectors
gsl_matrix *Eigvec_ptr = gsl_matrix_alloc (dimension, dimension);
// the workspace for gsl
gsl_eigen_symmv_workspace *worksp= gsl_eigen_symmv_alloc (dimension);

// Load the Hamiltonian matrix pointed to by Hmat_ptr
// The elements are labeled from 1 to N-1, but stored from 0 to N-2
for (int i = 1; i <= dimension; i++)
{
    for (int j = 1; j <= dimension; j++)
    {
        double Hij;
        if (i == j)                // diagonal matrix element
        {
            double r = double(i)*h;
            Hij = 2./hsq + V_ho(r);
        }
        else if (i == j+1)         // just above the diagonal
        {
            Hij = -1./hsq;
        }
        else if (i == j-1)        // just below the diagonal
        {
            Hij = -1./hsq;
        }
        else                       // all the other elements
        {
            Hij = 0.;
        }

        gsl_matrix_set (Hmat_ptr, i-1, j-1, Hij); // set the i-1,j-1 element
    }
}

// Find the eigenvalues and eigenvectors of the real, symmetric
// matrix pointed to by Hmat_ptr. It is partially destroyed
// in the process. The eigenvectors are pointed to by
// Eigvec_ptr and the eigenvalues by Eigval_ptr.
gsl_eigen_symmv (Hmat_ptr, Eigval_ptr, Eigvec_ptr, worksp);

// Sort the eigenvalues and eigenvectors in ascending order
gsl_eigen_symmv_sort (Eigval_ptr, Eigvec_ptr, GSL_EIGEN_SORT_VAL_ASC);

// Print out the results
// Allocate a pointer to one of the eigenvectors of the matrix
gsl_vector *eigenvector_ptr = gsl_vector_alloc (dimension);
for (int i = 1; i <= dimension; i++)
{
    double eigenvalue = gsl_vector_get (Eigval_ptr, i-1);
    gsl_matrix_get_col (eigenvector_ptr, Eigvec_ptr, i-1);

    cout << "eigenvalue " << i << " = "
         << scientific << eigenvalue << endl;
}

```

```

// Print out the eigenvector with the lowest eigenvalue to a file
if (i == 1)
{
    ofstream eigout ("eigen_tridiagonal.dat"); // open an output file
    eigout << "#3D harmonic oscillator" << endl;
    eigout << "#eigenvalue = " << scientific << eigenvalue << endl;
    eigout << endl << "# r u(r)" << endl;
    for (int j = 1; j <= dimension; j++)
    {
        eigout << fixed << double(j)*h << " "
            << scientific << gsl_vector_get (eigenvector_ptr, j-1) << endl;
    }
    eigout.close(); // close the output stream
}
}

// free the space used by the vector and matrices and workspace
gsl_matrix_free (Eigvec_ptr);
gsl_vector_free (Eigval_ptr);
gsl_matrix_free (Hmat_ptr);
gsl_vector_free (eigenvector_ptr);
gsl_eigen_symmv_free (worksp);

return (0); // successful completion
}

//*****
//***** V_ho *****
//
// Harmonic oscillator potential with k = 1/2.
// With m = 1/2 and hbar = 1, this means the energies
// are 3/2, 5/2, 7/2, ...
//
//*****
double
V_ho (double r)
{
    double k = 1./2.;
    return (k*r*r/2.);
}

//*****

```