

Jan 25, 11 21:36 **eigen_tridiagonal_class.cpp** Page 1/2

```

// file: eigen_tridiagonal_class.cpp
//
// Program to find bound state eigenvalues for a potential by
// discretizing and then diagonalizing the Hamiltonian in
// coordinate representation using a Hamiltonian class that
// is a "wrapper" for the GSL eigenvalue/eigenvector routines.
//
// Programmer: Dick Furnstahl furnstahl.1@osu.edu
//
// Revision history:
// 01/25/09 modified eigen_tridiagonal.cpp to move the GSL
// dependent parts to a Hamiltonian class.
//
// 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  $n = 0, 1, 2, 3, \dots$ 
//
//*****
// include files
#include <iostream>           // note that .h is omitted
#include <iomanip>           // note that .h is omitted
#include <fstream>
#include <cmath>
using namespace std;

#include "GslHamiltonian.h" // include the Hamiltonian class definitions

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
    int N = 0;
    cout << "Enter the number of steps (N): ";
    cin >> N;

    // Calculate  $h = \Delta r$ 
    double h = Rmax/double(N);
    double hsq = sqr(h);

    // The matrix dimension is N-1 (see the notes).
    int dimension = N-1;

    // Create the Hamiltonian object called my_hamiltonian
    Hamiltonian my_hamiltonian(dimension);

    // Load the Hamiltonian matrix
    for (int i = 1; i <= dimension; i++)
    {
        for (int j = 1; j <= dimension; j++)
        {
            double Hij;
            if (i == j)                // diagonal matrix element

```

Jan 25, 11 21:36 **eigen_tridiagonal_class.cpp** Page 2/2

```

        {
            double r = double(i)*h; // radial coordinate
            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.;
        }

        my_hamiltonian.set_element(i,j,Hij); // set the i,j element to Hij
    }
}

// Find eigenvalues and eigenvectors in ascending order
my_hamiltonian.find_eigenstuff();

// Print out the results
for (int i = 1; i <= dimension; i++)
{
    double eigenvalue = my_hamiltonian.get_eigenvalue(i);

    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 << my_hamiltonian.get_eigenvector(i,j) << endl;
        }
        eigout.close(); // close the output stream
    }
}

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  $3/2, 7/2, 11/2, \dots$ 
//
//*****
double
V_ho (double r)
{
    double k = 1./2.;
    return (k*r*r/2.);
}
//*****

```

```

Jan 30, 09 0:52      GslHamiltonian.h      Page 1/1
// file: GslHamiltonian.h
//
// Header file for the GslHamiltonian C++ class.
//
// Programmer: Dick Furnstahl  furnstahl.1@osu.edu
//
// Revision history:
//   01/25/09  original version
//
// Notes:
// * We encapsulate GSL matrix eigenvalue functions in this class
// * The GSL header file is included in GslHamiltonian.cpp and
//   anywhere we want to create GslHamiltonian objects.
// * Based on the documentation for the GSL library under
//   "Eigensystems" and on Chap. 15 of "Computational Physics"
//   by Landau and Paez.
// * See the GSL documentation for matrix, vector structures
// * 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_eigensymm
//   * 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:
// * check that find_eigenstuff has been run before get_eigenvalue
//   and get_eigenvector.
// * make get_eigenvector more efficient
//
//*****
// The ifndef/define macro ensures that the header is only included once
#ifndef GSLHAMILTONIAN_H
#define GSLHAMILTONIAN_H

// include files
#include <gsl/gsl_eigen.h>      // include the appropriate GSL header file(s)

class Hamiltonian
{
public:
    Hamiltonian (const int dim); // constructor
    ~Hamiltonian (); // destructor

    // accessor functions
    void set_element(const int i, const int j, double value);
    void find_eigenstuff();
    double get_eigenvalue(const int i);
    double get_eigenvector(const int i, const int j);

private:
    int dimension; // the matrix dimension
    gsl_matrix *Hmat_ptr; // gsl matrix with Hamiltonian
    gsl_vector *Eigval_ptr; // gsl vector with eigenvalues
    gsl_matrix *Eigvec_ptr; // gsl matrix with eigenvectors
    gsl_eigen_symmv workspace *worksp; // the workspace for gsl
    gsl_vector *eigenvector_ptr; // one of the eigenvectors
};

#endif

```

```

Jan 30, 09 0:52      GslHamiltonian.cpp      Page 1/1
// file: GslHamiltonian.cpp
//
// Definitions for the GslHamiltonian C++ class.
//
// Programmer: Dick Furnstahl  furnstahl.1@osu.edu
//
// Revision history:
//   01/25/09  Original version using eigen_tridiagonal.cpp as a guide.
//
//*****
// include files
#include <gsl/gsl_eigen.h>      // include the appropriate GSL header file
#include "GslHamiltonian.h"    // include the header for this class
//*****

Hamiltonian::Hamiltonian (const int dim) // Constructor for Hamiltonian
{
    dimension = dim;

    // Allocate space for the vectors, matrices, and workspace
    Hmat_ptr = gsl_matrix_alloc (dimension, dimension); // Hamiltonian
    Eigval_ptr = gsl_vector_alloc (dimension); // eigenvalue vector
    Eigvec_ptr = gsl_matrix_alloc (dimension, dimension); // eigenvector matrix
    worksp = gsl_eigen_symmv_alloc (dimension); // workspace
    eigenvector_ptr = gsl_vector_alloc (dimension); // one eigenvector
}

Hamiltonian::~Hamiltonian () // Destructor for Hamiltonian
{
    // 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);
}

void Hamiltonian::set_element(const int i, const int j, const double value)
{
    // The i,j element of the matrix is stored as the i-1,j-1 element of
    // the GSL matrix
    gsl_matrix_set (Hmat_ptr, i-1, j-1, value);
}

void Hamiltonian::find_eigenstuff()
{
    // 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 by default
    gsl_eigen_symmv_sort (Eigval_ptr, Eigvec_ptr, GSL_EIGEN_SORT_VAL_ASC);
}

double Hamiltonian::get_eigenvalue(int i)
{
    return gsl_vector_get (Eigval_ptr, i-1);
}

double Hamiltonian::get_eigenvector(int i, int j)
{
    gsl_matrix_get_col (eigenvector_ptr, Eigvec_ptr, i-1);
    return gsl_vector_get (eigenvector_ptr, j-1);
}

//*****

```