// file: eigen_basis_class.cpp

Program to find bound state eigenvalues for various potentials
by diagonalizing the Hamiltonian in a truncated harmonic
oscillator basis using a Hamiltonian class that
is a "wrapper" for the GSL eigenvalue/eigenvector routines.

Programmer: Dick Furnstahl furnstahl.1@osu.edu

Revision history:
01/28/09 modified eigen_basis.cpp to move the GSL
dependent parts to a Hamiltonian class.

Notes:
* Had to re-index from 1 instead of from 0
* We use gsl integration qagiu for the integrals from
  0 to Infinity (calculating matrix elements of H).
* Start with l=0 (and generalize later)

To do:
* Add the Morse potential (function is given but not incorporated)
* Generalize to l>0.
* Add the Morse potential (function is given but not incorporated)

To do:
* Start with l=0 (and generalize later)
  0 to Infinity (calculating matrix elements of H).
* We use gls_integration_qagiu for the integrals from
* Had to re

Notes:
* dependent parts to a Hamiltonian class.

Revision history:

is a "wrapper" for the GSL eigenvalue/eigenvector routines.

Program to find bound state eigenvalues for various potentials
by diagonalizing the Hamiltonian in a truncated harmonic
oscillator basis using a Hamiltonian class that
is a "wrapper" for the GSL eigenvalue/eigenvector routines.

Programmer: Dick Furnstahl furnstahl.1@osu.edu

Revision history:
01/28/09 modified eigen_basis.cpp to move the GSL
dependent parts to a Hamiltonian class.

Notes:
* Had to re-index from 1 instead of from 0
* We use gsl integration qagiu for the integrals from
  0 to Infinity (calculating matrix elements of H).
* Start with l=0 (and generalize later)

To do:
* Add the Morse potential (function is given but not incorporated)
* Generalize to l>0.
* Add the Morse potential (function is given but not incorporated)

To do:
* Start with l=0 (and generalize later)
  0 to Infinity (calculating matrix elements of H).
* We use gls_integration_qagiu for the integrals from
* Had to re

Notes:
* dependent parts to a Hamiltonian class.

Revision history:

is a "wrapper" for the GSL eigenvalue/eigenvector routines.

Program to find bound state eigenvalues for various potentials
by diagonalizing the Hamiltonian in a truncated harmonic
oscillator basis using a Hamiltonian class that
is a "wrapper" for the GSL eigenvalue/eigenvector routines.

Programmer: Dick Furnstahl furnstahl.1@osu.edu

Revision history:
01/28/09 modified eigen_basis.cpp to move the GSL
dependent parts to a Hamiltonian class.

Notes:
* Had to re-index from 1 instead of from 0
* We use gsl integration qagiu for the integrals from
  0 to Infinity (calculating matrix elements of H).
* Start with l=0 (and generalize later)

To do:
* Add the Morse potential (function is given but not incorporated)
* Generalize to l>0.
* Add the Morse potential (function is given but not incorporated)

To do:
* Start with l=0 (and generalize later)
  0 to Infinity (calculating matrix elements of H).
* We use gls_integration_qagiu for the integrals from
* Had to re

Notes:
* dependent parts to a Hamiltonian class.

Revision history:

is a "wrapper" for the GSL eigenvalue/eigenvector routines.

Program to find bound state eigenvalues for various potentials
by diagonalizing the Hamiltonian in a truncated harmonic
oscillator basis using a Hamiltonian class that
is a "wrapper" for the GSL eigenvalue/eigenvector routines.

Programmer: Dick Furnstahl furnstahl.1@osu.edu

Revision history:
01/28/09 modified eigen_basis.cpp to move the GSL
dependent parts to a Hamiltonian class.

Notes:
* Had to re-index from 1 instead of from 0
* We use gsl integration qagiu for the integrals from
  0 to Infinity (calculating matrix elements of H).
* Start with l=0 (and generalize later)

To do:
* Add the Morse potential (function is given but not incorporated)
* Generalize to l>0.
* Add the Morse potential (function is given but not incorporated)

To do:
* Start with l=0 (and generalize later)
  0 to Infinity (calculating matrix elements of H).
* We use gls_integration_qagiu for the integrals from
* Had to re

Notes:
* dependent parts to a Hamiltonian class.

Revision history:

is a "wrapper" for the GSL eigenvalue/eigenvector routines.

Program to find bound state eigenvalues for various potentials
by diagonalizing the Hamiltonian in a truncated harmonic
oscillator basis using a Hamiltonian class that
is a "wrapper" for the GSL eigenvalue/eigenvector routines.

Programmer: Dick Furnstahl furnstahl.1@osu.edu

Revision history:
01/28/09 modified eigen_basis.cpp to move the GSL
dependent parts to a Hamiltonian class.

Notes:
* Had to re-index from 1 instead of from 0
* We use gsl integration qagiu for the integrals from
  0 to Infinity (calculating matrix elements of H).
* Start with l=0 (and generalize later)

To do:
* Add the Morse potential (function is given but not incorporated)
* Generalize to l>0.
* Add the Morse potential (function is given but not incorporated)

To do:
* Start with l=0 (and generalize later)
  0 to Infinity (calculating matrix elements of H).
* We use gls_integration_qagiu for the integrals from
* Had to re

Notes:
* dependent parts to a Hamiltonian class.

Revision history:

is a "wrapper" for the GSL eigenvalue/eigenvector routines.

Program to find bound state eigenvalues for various potentials
by diagonalizing the Hamiltonian in a truncated harmonic
oscillator basis using a Hamiltonian class that
is a "wrapper" for the GSL eigenvalue/eigenvector routines.

Programmer: Dick Furnstahl furnstahl.1@osu.edu

Revision history:
01/28/09 modified eigen_basis.cpp to move the GSL
dependent parts to a Hamiltonian class.

Notes:
* Had to re-index from 1 instead of from 0
* We use gsl integration qagiu for the integrals from
  0 to Infinity (calculating matrix elements of H).
* Start with l=0 (and generalize later)

To do:
* Add the Morse potential (function is given but not incorporated)
* Generalize to l>0.
* Add the Morse potential (function is given but not incorporated)

To do:
* Start with l=0 (and generalize later)
  0 to Infinity (calculating matrix elements of H).
* We use gls_integration_qagiu for the integrals from
* Had to re

Notes:
* dependent parts to a Hamiltonian class.

Revision history:

is a "wrapper" for the GSL eigenvalue/eigenvector routines.

Program to find bound state eigenvalues for various potentials
by diagonalizing the Hamiltonian in a truncated harmonic
oscillator basis using a Hamiltonian class that
is a "wrapper" for the GSL eigenvalue/eigenvector routines.

Programmer: Dick Furnstahl furnstahl.1@osu.edu

Revision history:
01/28/09 modified eigen_basis.cpp to move the GSL
dependent parts to a Hamiltonian class.

Notes:
* Had to re-index from 1 instead of from 0
* We use gsl integration qagiu for the integrals from
  0 to Infinity (calculating matrix elements of H).
* Start with l=0 (and generalize later)

To do:
* Add the Morse potential (function is given but not incorporated)
* Generalize to l>0.
* Add the Morse potential (function is given but not incorporated)

To do:
* Start with l=0 (and generalize later)
  0 to Infinity (calculating matrix elements of H).
* We use gls_integration_qagiu for the integrals from
* Had to re

Notes:
* dependent parts to a Hamiltonian class.

Revision history:

is a "wrapper" for the GSL eigenvalue/eigenvector routines.

Program to find bound state eigenvalues for various potentials
by diagonalizing the Hamiltonian in a truncated harmonic
oscillator basis using a Hamiltonian class that
is a "wrapper" for the GSL eigenvalue/eigenvector routines.

Programmer: Dick Furnstahl furnstahl.1@osu.edu

Revision history:
01/28/09 modified eigen_basis.cpp to move the GSL
dependent parts to a Hamiltonian class.

Notes:
* Had to re-index from 1 instead of from 0
* We use gsl integration qagiu for the integrals from
  0 to Infinity (calculating matrix elements of H).
* Start with l=0 (and generalize later)

To do:
* Add the Morse potential (function is given but not incorporated)
* Generalize to l>0.
* Add the Morse potential (function is given but not incorporated)

To do:
* Start with l=0 (and generalize later)
  0 to Infinity (calculating matrix elements of H).
* We use gls_integration_qagiu for the integrals from
* Had to re

Notes:
* dependent parts to a Hamiltonian class.

Revision history:

is a "wrapper" for the GSL eigenvalue/eigenvector routines.
double Hij (hij_parameters ho_parameters)
{
  gsl_integration_workspace *work = gsl_integration_workspace_alloc (1000);
  gsl_function F_integrand;

  // calculate the i'th-j'th matrix element of the Hamiltonian
  // in a Harmonic oscillator basis. 
  // This routine just passes
  // the integrand Hij_integrand to a GSL integration routine
  (gsl_integration_qagiu that integrates it over r from 0
  to infinity)

  // Take l=0 only for now

  // define variables for debugging 2nd derivative
  double n_i, n_j;

  // set up the integrand
  F_integrand.function = &Hij_integrand;
  F_integrand.params = params_ptr;

  // start integral from 0 (to infinity)
  double lower_limit = 0.;

  // avoid round-off problems
  double abs_error = 1.0e-8;

  // the result will usually be much better
  double rel_error = 1.0e-8;

  // the estimated error from the integration
  double error = 0.;

  // send back the result of the integration
  // we'll pass i, j, mass, b_ho
  params_ptr = ho_parameters;

  // set up the integrand
  F_integrand.function = &Hij_integrand;
  F_integrand.params = params_ptr;

  // carry out the integral over r from 0 to infinity
  gsl_integration_qagiu (&F_integrand, lower_limit, abs_error, rel_error, 1000, work, &result, &error);

  // eventually we should do something with the error estimate
  switch (potential_index)
  {
    default:
      cout << "Shouldn't get here!\n"
      return (1);
    break;
    case 1:
      // coulomb
      Zeq = 1.;
      potl_params.param1 = Zeq;
      return (ho_radial (n_i, l, b_ho, x)
        * (ho_eigenvalue (n_j, l, b_ho, mass)
          + V_coulomb (x, &potl_params))
        * ho_radial (n_j, l, b_ho, x));
      break;
    case 2:
      // square well
      R = 1.;
      V0 = 50.;
      potl_params.param1 = V0;
      potl_params.param2 = R;
      return (ho_radial (n_i, l, b_ho, x)
        * (ho_eigenvalue (n_j, l, b_ho, mass)
          + V_square_well (x, &potl_params))
        * ho_radial (n_j, l, b_ho, x));
      break;
    }

  return (result);
}

double potential (double x, void *params_ptr)
{
  potential_parameters potl_params;

  // square well
  if (potl_params.param2 > potl_params.param1)
    return (V_square_well (x, &potl_params));

  // coulomb
  if (potl_params.param1 > 0.)
    return (V_coulomb (x, &potl_params));

  // hydrogen-like
  return (ho_radial (n_i, l, b_ho, x));
}

int main ()
{
  double mass, b_ho;
  double hbar = 1.;
  double omega;
  double ho_pot;

  // value of ho potential at current x

  // define variables for debugging 2nd derivative
  double h = 0.01;

  // debugging code to calculate 2nd derivative by hand
  double fp, f, fm, deriv2;

  // debugging code to use crude 2nd derivative
  deriv2 = -(fp - f) - (f - fm) / (h * h) / (2. * mass);

  // set up the potential according to potential index
  switch (potential_index)
  {
    default:
      return (f);
    break;
  }

  return (deriv2);
}

void deriv2 (double x, void *params_ptr)
{
  potential_parameters potl_params;

  // square well
  if (potl_params.param2 > potl_params.param1)
    return (V_square_well (x, &potl_params));

  // coulomb
  if (potl_params.param1 > 0.)
    return (V_coulomb (x, &potl_params));

  // hydrogen-like
  return (ho_radial (n_i, l, b_ho, x));
}
double V_coulomb (double r, potential_parameters * potl_params_ptr)
{
    double Zesq = potl_params_ptr->param1;
    return (-Zesq / r);
}

//****************************************************************************
//**************************** V_square_well ****************************
// Square well potential of radius R and depth V0
//****************************************************************************
double V_square_well (double r, potential_parameters * potl_params_ptr)
{
    double V0 = potl_params_ptr->param1;
    double R = potl_params_ptr->param2;
    if (r < R)
    {
        return (-V0);  // inside the well of depth V0
    }
    else
    {
        return (0.);  // outside the well
    }
}

//****************************************************************************
//************************** V_morse ***********************************
// Morse potential with equilibrium bond length r_eq and potential energy for bond formation D_eq
//****************************************************************************
double V_morse (double r, potential_parameters * potl_params_ptr)
{
    double D_eq = potl_params_ptr->param1;
    double r_eq = potl_params_ptr->param2;
    return ( D_eq * sgr(1. - exp(-abs(r-r_eq))) );
}