Neutron Drop

Generated by Doxygen 1.8.6

Wed Aug 3 2016 16:47:11

Contents

Chapter 1

Installation and Instructions For Use

1.1 Introduction

Solves the problem of n neutrons in a harmonic oscillator trap by a self-consistent, Hartree-Fock calculation.

1.2 Installation

The source code for `HFSolver` includes the following Fortran modules:

- `HFSolver_main.f90`
- `HF_HartreeFock.f90`
- `HF_HOQuadrature.f90`
- `HF_LDA.f90`
- `HF_Minnesota.f90`
- `HF_variables.f90`
- `types.f90`

as well as one input file, `HF_input.dat` (to be described in the following section).

To compile, simply type `make`. The executable is called `HFSolver`.

1.3 Instructions For Use

1.3.1 HF_input.dat

A template input file is included. No default arguments are provided by the code, so you MUST specify every item in the output file and in the order they are given in the template; otherwise the code will not run. Furthermore, the input file is case- sensitive. The individual entries are explained below:

1.3.1.1 Number of particles

Set the number of neutrons in the trap

1.3.1.2 Maximum number of iterations

Set the maximum number of iterations

1.3.1.3 Fermi Momentum

Specify the value of the Fermi momentum k_F (in fm^{-1}) to be used in LDA and DME calculations. Typical values are between 1 and 2.

1.3.1.4 Orbitals file

List the name of the input file in which you index your m-scheme orbitals. If they were generated using Morten's `vNN` code, the file will probably be called `spM.dat`. The file should have the following format:

```
Legend:  n l 2j 2mj 2tz
Orbit number:  1 0 0 1 -1 -1
Orbit number:  2 0 0 1 1 -1
Orbit number:  3 0 0 1 -1 1
Orbit number:  4 0 0 1 1 1
...
```

1.3.1.5 Matrix elements file

List the name of the input file in which you list your two-body m-scheme matrix elements. If they were generated using Morten's `vNN` code, the file will probably be called `VM-scheme.dat`. The file should have the following format:

```
Legend:  n l 2j 2mj 2tz
 Matrix elements < ab | V | cd> in mscheme
 Legend:  a b c d and < ab | V | cd>
 5 6 5 13 0.113470E+01
 5 6 5 22 -.287678E+00
 5 6 5 29 0.406838E+00
 5 6 5 38 -.659420E-01
...
```

where a, b, c, d refer to the orbitals indexed in **Orbitals file**

1.3.1.6 Output file

Specify the name of your output file.

1.3.1.7 density_file

Specify the name of the density file, which will contain the density distribution on a format suitable for plotting utilities such as gnuplot.

1.3.1.8 Type of calculation

If the option `truncated` is selected, the program assumes $l = 0$ and calculates the two-body matrix elements exactly using the Minnesota potential (using the parameters given in HF_truncated_v2.pdf).

If the option `spherical` is selected, the program will read the matrix elements from a file, **Matrix elements file**. There is (in principle) no limitation on l in this case.

If the option `LDA` is selected, the program will use the Local Density Approximation to the Hamiltonian.

If the option `DME` is selected, the program will use the Density Matrix Expansion to approximate the Hamiltonian.

1.3.1.9 N_max

Specify the maximum value of the orbital quantum number n. This is only applicable using the option `truncated`. All other options ignore this number.

Chapter 2

Data Type Index

2.1 Data Types List

Here are the data types with brief descriptions:

Chapter 3

Data Type Documentation

3.1 hartreefock Module Reference

Contains functions and subroutines for the Hartree-Fock self- consistent part of the calculation, include building the density matrix $\rho_{\mu\nu}$ and the single particle potential $\Gamma_{\alpha\beta}$, and diagonalizing the Hamiltonian to extract its eigenvectors and eigenvalues (see pg. 2 of HF_truncated_v2.pdf).

Public Member Functions

- subroutine **read_input** ()
- subroutine initialize_hf

  Allocates the arrays which will be used in the Hartree-Fock calculation, and initializes the matrix $D_{\mu i} = \delta\mu i$ (see eqns. 1, 5 of HF_truncated_v2.pdf).
- subroutine construct_rho

  Constructs the density matrix $\rho_{\mu\nu} = \sum_{i=1}^{N} D_{\mu i} D_{\nu i}^{*}$.
- subroutine construct_gamma

  Constructs the single-particle potential $\Gamma_{\alpha\beta} = \sum_{\mu\nu} v_{\alpha\nu\beta\mu}\rho\mu\nu$.
- subroutine diagonalize_h

  Diagonalizes a matrix, returning its eigenvalues and eigenvectors, by calling the LAPACK routine dsyev. Unless I'm mistaken, eigenvalues are returned in the array `E_Values`, and eigenvectors in the array `D_mat`.
- real(dp) function trace_product (A, B)

  Computes the trace of the matrix produce $Tr(AB)$.
- real(dp) function trace (A)

  Computes the trace of a square matrix $Tr(A)$.

3.1.1 Detailed Description

Contains functions and subroutines for the Hartree-Fock self- consistent part of the calculation, include building the density matrix $\rho_{\mu\nu}$ and the single particle potential $\Gamma_{\alpha\beta}$, and diagonalizing the Hamiltonian to extract its eigenvectors and eigenvalues (see pg. 2 of HF_truncated_v2.pdf).

The documentation for this module was generated from the following file:

- HF_HartreeFock.f90

3.2 ho_quadrature Module Reference

Contains the functions and subroutines needed for defining the Laguerre polynomials, which describe the radial part of the harmonic oscillator wave function (which is our starting basis). Also contains functions and subroutines for evaluating integrals of these and other functions, using Gauss-Laguerre quadrature.

Public Member Functions

- subroutine gausslaguerrewx (alfa, w, x)

  Gauss-Laguerre quadrature routine. The w(i) are the weights and the x(i) are the corresponding roots.
- subroutine laguerrel (n, alpha, x, Ln, Lnp, Lnm1)

  Evaluates the Laguerre polynomial $L_n^{\alpha}(x)$ using the recurrence relation (see section 1.2 of HO_basis.pdf). Returns `Ln`*. Optionally, the user may include a fifth and sixth argument, in which case the subroutine will also return the derivative $L_n' \rightarrow$* `Lnp` *and the previous Laguerre polynomial $L_{n-1} \rightarrow$* `Lnm1`*.*
- subroutine radialhoall (n_max, l_max, xi, b, Rnl)

  Similar to `LaguerreAll`*, this subroutine computes and stores an array* `Rnl` *in which* `Rnl(0,n,l)` *is the radial part of the 3D harmonic oscillator wave function R_{nl},* `Rnl(1,n,l)` *is its first derivative, and* `Rnl(0,n,l)` *is its second derivative. See eqn. 5 of HO_basis.pdf.*
- subroutine normalizationall (n_max, l_max, Anl)

  Calls the function `HO_Normalization(n,l)` *for all values of n,l up to n_{max},l_{max}, and stores the result to an array* `Anl(n,l)`*.*
- subroutine laguerreall (n_max, l_max, x, Ln)

  Similar to `LaguerreL`*, except that this computes ALL Laguerre polynomials up to* `n_max, l_max` *using the recurrence relation (see section 1.2 of HO_basis.pdf).*
- real(dp) function factrl (n)

  Numerically evaluates the value of the factorial function, $n! = n(n-1)(n-2)....$
- real(dp) function doublefactrl (n)

  Numerically evaluates the value of the double-factorial function, $n!! = n(n-2)(n-4)....$
- real(dp) function ho_normalization (n, l)

  Evaluates the normalization prefactor for the 3D harmonic oscillator, as written in eqn. 5 and 6 of HO_basis.pdf, for a given value of n and l.

3.2.1 Detailed Description

Contains the functions and subroutines needed for defining the Laguerre polynomials, which describe the radial part of the harmonic oscillator wave function (which is our starting basis). Also contains functions and subroutines for evaluating integrals of these and other functions, using Gauss-Laguerre quadrature.

3.2.2 Member Function/Subroutine Documentation

3.2.2.1 real(dp) function ho_quadrature::doublefactrl (integer, intent(in) *n*)

Numerically evaluates the value of the double-factorial function, $n!! = n(n-2)(n-4)....$

Parameters

in	*n*	An integer

The documentation for this module was generated from the following file:

- HF_HOQuadrature.f90

3.3 lda Module Reference

Contains functions and subroutines which allow you to approximate the potential using the Local-Density Approximation or the Density Matrix Expansion.

Public Member Functions

- real(dp) function rho_lda (r)

  Computes the density in the coordinate basis in the Local Density Approximation (see eqn. 60 of HF_extensions.pdf).
- subroutine calculate_gamma_lda

  Calls the function `gamma_LDA(n,n',l)` *to calculate the matrix elements* Γ_{ij},Γ_{ji} *(see HF_Extensions.pdf, eqn. 58).*
- real(dp) function gamma_lda (n, np, l)

  Performs the integral in HF_Extensions.pdf, eqn. 58. `(n, np, l)` $= n,n',l$.
- real(dp) function integralvc ()

  Evaluates the integral in HF_Extensions.pdf eqn. 54.
- real(dp) function sphericalbesselj1 (x)

  Spherical Bessel function $J_1(x)$.
- real(dp) function sphericalbesselj3 (x)

  Spherical Bessel function $J_3(x)$.
- subroutine plot_rho_lda

  Writes r *and* $\rho_{LDA}(r)$ *(or rather,* $r^2\rho_{LDA}(r)$*) to a file* `mixed_rho_plot**.dat` *for plotting.*
- real(dp) function trace_rho_lda ()

  Computes the trace of the density matrix in LDA by integrating the density over coordinate space. If everything is working properly, this integral should return the number of particles.
- subroutine sample_rho_lda

  Calls upon `rho_LDA` *at an array of points defined by the Gauss- Laguerre quadrature mesh, and stores the result in a global array* `rho_LDA` *to be used by* `gamma_LDA` *to calculate integral 58 of HF_extensions.pdf.*
- subroutine dme_fields (r, rho, tau, del_rho)

  Calculates, at an arbitrary point R*, the fields* $\rho(R)$*,* $\tau(R) = \nabla_1 \cdot \nabla_2 \rho(r_1 r_2)|_{r_1=r_2=R}$ *and* $\nabla\rho(r)$*. The subroutine* `sample_DME_fields` *will call this subroutine for each point in the integration/quadrature mesh.*
- real(dp) function gamma_dme (n, np, l)

  Performs the integration (using Guass-Laguerre quadrature with weight function w) in HF_extensions.pdf eqn. 58, except using DME instead of LDA. `(n, np, l)` $= n,n',l$.
- subroutine sample_dme_fields

  Calls upon `DME_fields` *at an array of points defined by the Gauss- Laguerre quadrature mesh, and stores the result in the global arrays* `rho_quad`*,* `tau_quad`*, and* `delrho_quad` *to be used by* `gamma_DME` *to calculate integral 58 of HF_extensions.pdf.*
- subroutine calculte_couplings

  Here the coupling constants $C^{\rho\rho}, C^{\rho\tau}$*, and* $C^{\rho\nabla^2\rho}$ *are calculated using the kernels defined in the functions* `C_rhorho_kernel(r)` *and* `C_rhotau_kernel(r)` *(the kernel of the* $C^{\rho\nabla^2\rho}$ *term is equal to* $-\frac{C^{\rho\tau}}{4}$*; see HF_extensions.pdf eqn. 63).*
- real(dp) function c_rhorho_kernel (r)

  Contains the kernel of the integral used to compute $C^{\rho\rho}$*. Essentially, it is everything inside the integral in eqn. 64 of HF_extensions.pdf, except the integral has been transformed into a form that permits it to be evaluated using our Gauss-Laguerre quadrature scheme.*
- real(dp) function c_rhotau_kernel (r)

  Contains the kernel of the integral used to compute $C^{\rho\tau}$ *(see eqn. 63 of HF_extensions.pdf) As in the case of* $C^{\rho\rho}$*, the integral has been transformed into a form that permits it to be evaluated using our Gauss-Laguerre quadrature scheme.*
- subroutine plot_dme_fields

  Writes r *and* $\rho_{DME}(r)$ *(or rather,* $r^2\rho_{DME}(r)$*) to a file* `dmefields_surface_sat.dat` *for plotting.*
- real(dp) function rms_dme ()

  Computes the RMS radius of the neutron drop in the DME approximation.

3.3.1 Detailed Description

Contains functions and subroutines which allow you to approximate the potential using the Local-Density Approximation or the Density Matrix Expansion.

The documentation for this module was generated from the following file:

- HF_LDA.f90

3.4 minnesota Module Reference

The interaction part of the potential can be calculated relatively simply using the Minnesota potential for $l = 0$ (using the subroutine `calculate_TBME`), or you can read in the m-scheme matrix elements from a file generated by Morten's `vNN` code. Finally, you can estimate the potential using the Local-Density Approximation or the Density Matrix Expansion (although that capability resides in a separate module, `HF_LDA.f90`).

Public Member Functions

- subroutine read_orbitals

  Reads the file `spM.dat` generated by Morten's `vNN` code and stores, for each particle, the quantum numbers n, h, j, m, and the isospin projection. Currently it is configured to flag neutron states (by setting `hf_flag(i)=1`). Since we won't need every state in the list for a calculation, the array `HO_inverse` is used to match the index j (referred to by the m-scheme two-body term $V(j_1, j_2, j_3, j_4)$ - see RHS of eq. 4 of HF_fullspherical.pdf) with the index of the corresponding state in the `spM.dat` list(indexed by i).
- subroutine read_tbme

  Reads the `VM-scheme.dat` file generated by Morton's code. The first four columns refer back to specific single-particle states indexed in `spM.dat`, and the final column gives the corresponding two-body matrix element. It uses the `ho_flag` array to import only neutron states. It returns the m, m'-averaged two- body matrix element (eq. 4 of HF_fullspherical.pdf).
- subroutine initialize_minnseota

  Initializes the one-body part of the Hamiltonian, that is, the first term $t_{\alpha\beta}$ in eq. 2 of HF_truncated_v2.pdf. The matrix is diagonal in the harmonic oscillator basis (which we start from), and its eigenvalues are given by $\hbar\omega\left(2n + l + \frac{3}{2}\right)$.
- integer function fermi_level ()

  Given the number of particles `Nparticles`, finds the number of occupied levels in the system.
- subroutine calculate_tbme

  Calls the function Minnesota_TBME to calculate the two-body matrix elements, and stores the result in an array such that $V_{i_1 i_2 i_3 i_4} = V_{i_1 i_2 i_4 i_3} = V_{i_2 i_1 i_3 i_4} = V_{i_2 i_1 i_4 i_3}$. Consequnetly, these matrix elements respect fermion antisymmetry (see HF_truncated_v2.pdf eqns. 17-21).
- real(dp) function minnesota_tbme (n1, n2, n3, n4)

  Using Gauss-Laguerre quadrature with weights w_i and w_j, performs the integral of the kernel created in `Minnesota_Kernel`, which, when normalized with the harmonic oscillator coefficients A_i, gives the value of the TBME matrix element $V_{n_1 n_2 n_3 n_4}$. `Ngauss` is the number of mesh points used for the Gauss-Laguerre quadrature; more than 100 seemed to lead to problems so we set it to 95. The case $\alpha = 0.5$ is specific to the case $l = 0$.
- real(dp) function minnesota_kernel (n1, n2, n3, n4, xi, xj)

  Sets up the kernel of the integral found in HF_truncated_v2.pdf eqn. 11 for the form of the Minnesota potential described in eqns. 26-29.

3.4.1 Detailed Description

The interaction part of the potential can be calculated relatively simply using the Minnesota potential for $l = 0$ (using the subroutine `calculate_TBME`), or you can read in the m-scheme matrix elements from a file generated by Morten's `vNN` code. Finally, you can estimate the potential using the Local-Density Approximation or the Density Matrix Expansion (although that capability resides in a separate module, `HF_LDA.f90`).

The documentation for this module was generated from the following file:

- HF_Minnesota.f90

3.5 types Module Reference

Defines the integer parameters **sp**, **dp** and **qp** to be used as kinds to define real variables as single precision (32-bit), double precision (64-bit) and quadruple precision (128-bit) (depending on the machine and compiler, this last one may not always be available).

Public Attributes

- integer, parameter sp = REAL32
  single precision kind
- integer, parameter dp = REAL64
  double precision kind
- integer, parameter qp = REAL128
  quadruple precision kind
- real(dp), parameter pi =acos(-1._dp)
  π *= 3.141592...*

3.5.1 Detailed Description

Defines the integer parameters **sp**, **dp** and **qp** to be used as kinds to define real variables as single precision (32-bit), double precision (64-bit) and quadruple precision (128-bit) (depending on the machine and compiler, this last one may not always be available).

The intrinsic module iso_fortran_env (Fortran 2008 and later) is used.

More types may be defined later (i.e. larger integers)

Fundamental constants (i.e. π, e, ...) may also be defined here if desired. This is a good place to define this type of constants as all modules and the main program will (in principle) use this module

Parameters

sp	single precision kind
dp	double precision kind
qp	quadruple precision kind (not available in every machine)
pi	$\pi = 3.141592\ldots$

Author

Rodrigo Navarro Perez

The documentation for this module was generated from the following file:

- types.f90

3.6 variables Module Reference

Public Attributes

- integer **nsize**
- integer **nparticles**
- integer **n_orbitals**

- integer **n_n_orbitals**
- integer **noccupied**
- integer **maxit**
- integer, parameter **n_quad** = 95
- real(dp), dimension(1:n_quad) **w_quad**
- real(dp), dimension(1:n_quad) **x_quad**
- real(dp), dimension(1:n_quad) **rho_quad**
- real(dp), dimension(1:n_quad) **tau_quad**
- real(dp), dimension(1:n_quad) **delrho_quad**
- real(dp), dimension(:,:), allocatable **d_mat**
- real(dp), dimension(:,:), allocatable **rho_mat**
- real(dp), dimension(:,:), allocatable **h_mat**
- real(dp), dimension(:,:), allocatable **t_mat**
- real(dp), dimension(:,:), allocatable **gamma_mat**
- real(dp), dimension(:,:,:,:), allocatable **v_mat**
- real(dp), dimension(:), allocatable **e_values**
- real(dp), dimension(:), allocatable **e_prev**
- real(dp) **small** = 1.e-4_dp
- real(dp) **k_fermi** = 3.01_dp
- real(dp) **delta_e**
- real(dp) **c_hartree**
- real(dp) **c_rhorho**
- real(dp) **c_rhotau**
- real(dp) **c_rhodelrho**
- real(dp) **trrho**
- integer, dimension(:), allocatable **n_ho**
- integer, dimension(:), allocatable **l_ho**
- integer, dimension(:), allocatable **ho_index**
- integer, dimension(:), allocatable **m_ho**
- integer, dimension(:), allocatable **j_ho**
- integer, dimension(:), allocatable **ho_flag**
- integer, dimension(:), allocatable **ho_inverse**
- integer, dimension(:), allocatable **tz_ho**
- integer, dimension(:), allocatable **n_hf**
- integer, dimension(:), allocatable **l_hf**
- integer, dimension(:), allocatable **j_hf**
- logical **calc_ivc**
- logical **calc_couplings**
- logical **truncated**
- logical **approximated_rho**
- character(30) **orbitals_file**
- character(30) **elements_file**
- character(30) **type_of_calculation**
- character(30) **output_file**
- character(30) **density_file**

The documentation for this module was generated from the following file:

- HF_variables.f90

Index