Input options for *hartreeFock*

- The **hartreeFock** program should be run as:
  - *./hartreeFock inputFile.in*
  - ``inputFile.in" in a plain-text input file, that contains all input options (if no input file is given, program looks for the default one, named "hartreeFock.in")
- First, the program reads in the main input options from the four main input "blocks" (Atom, Nucleus, HartreeFock, and Grid). It will use these to generate wavefunction/Hartree-Fock etc. Then, any number of optional "modules" are run using the above-calculated wavefunctions (e.g., these might calculate matrix elements, run tests etc.). The input blocks and options can be given in any order
- In general, the input file will have the following format:

```
Atom { <input options> }
Nucleus { <input options> }
Grid { <input options> }
HartreeFock { <input options> }
//Optional modules:
Module::firstModule { <input options> }
Module::secondModule { <input options> }
```

- Most options have a default, and can be left blank, explicitly set to 'default', or removed entirely. (A few inputs (e.g., Atom/Z, and HartreeFock/core) are required)
- The curly-braces are important. Don't use any other curly-braces (nested braces are not allowed)
- Input file uses c++ style line comments (not block comments). Any commented-out line will not be read. White-space is ignored.
- For example, the following four inputs are all equivalent

```
Atom {
  Z = Cs;
  A;
}
Atom {
  Z = Cs;
//  A;
}
Atom { Z = Cs; }
Atom{Z=Cs;A=default;}
```

- All available inputs for each input block are listed below
  - Inputs are taken in as either text, boolean (true/false), integers, or real numbers. (Don't use inverted comma's/quote marks)
  - These will be denoted by [t], [b], [i], [r]

Atom

```
Atom {
  Z = Cs;    //[t/i] required
  A;         //[i] Will look-up default value if default
  varAlpha2; //[r] default = 1
}
```

- Z: Which atom. Can enter as symbol (e.g., Z = Cs;) or integer (e.g., Z = 55;). Required (no default)
- A: nuclear mass number. Leave blank to look up default value
- varAlpha2: scale factor for (inverse) speed of light^2 [alpha in atomic un.]: alpha^2 = varAlpha2 * alpha_real^2.
  - default=1. put a very small number to achieve non-relativistic limit (useful for tests)

HartreeFock

```
HartreeFock {
  core = [Xe]; //[t] required
  valence;     //[t] default = none
  sortOutput;  //[b] default = true
  method;      //[t] default = HartreeFock
  convergence; //[r] default = 1.0e-12
  orthonormaliseValence; //[b] default = false
}
```

- core: Core configuration. Required (no default)
  - Format: [Noble gas],extra (comma separated, no spaces)
  - Can also enter entire configuration, e.g., 1s2,2s2,2p6,.... (As well as Noble gas, can use Zn,Cd,Hg,Cn)
  - Can also add negative values for occupations
  - E.g. :
    - Cs (V^N-1): '[Xe]'
    - Au (V^N-1): '[Xe],4f14,5d10' or '[Hg],6s-2'
    - Tl (V^N-1): '[Xe],4f14,5d10,6s2' or '[Hg]'
    - I (V^N): '[Cd],5p5' or '[Xe],5p-1'
    - H-like: enter as: [] (or 1s0) -- no electrons in core
- valence: which valence states to calculate
  - e.g., "7sp5df" will do s and p states up to n=7, and d and f up to n=5
- sortOutput: true or false. Sort output by energy.
- method: which method to use. can be:
  - HartreeFock(default), ApproxHF, Hartree, GreenPRM, TietzPRM
- convergence: level we try to converge to.
- orthonormaliseValence: true/false. Orthogonalise valence states? false by default. Only really for testing

Nucleus

```
Nucleus {
  type;    //[t] default = Fermi
  rrms;    //[r] will loop-up default value based on Z,A
  skin_t;  //[r] default = 2.3
}
//nb: all of these are optional, hence entire block can be omitted
```

- rrms: nuclear root-mean-square charge radius (in femptometres = 10^-15m)
- type: Which distribution to use for nucleus? Options are: Fermi (default), spherical, point
- skin_t: skin thickness [only used by Fermi distro]

Grid

```
Grid {
  r0;         //[r] default = 1.0e-5
  rmax;       //[r] default = 150.0
  num_points; //[i] default = 1600
  type;       //[t] default = loglinear
  b;          //[r] default = 4.0
  fixed_du;   //[r] default = -1. du>0 means calculate num_points
}
```

- r0: grid starting point (in atomic units)
- rmax: Final grid point (in atomic units)
- num_points: number of points in the grid
- type: What type of grid to use? options are: loglinear (default), logarithmic, linear
  - Note: 'linear' grid requires a *very* large number of points to work, and should essentially never be used.
- b: only used for loglinear grid; the grid is roughly logarithmic below this value, and linear after it. Default is 4.0 (atomic units). If b<0 or b>rmax, will revert to using a logarithmic grid

Modules and MatrixElements

Modules and MatrixElements work in essentially the same way. Each MatrixElements/Modules block will be run in order. You can comment-out just the block name, and the block will be skipped.

MatrixElements blocks calculate reduced matrix elements of given operator, Modules can do anything.

For MatrixElements, there are some options that apply for any operator; and then there are some options specific to each operator

```
MatrixElements::ExampleOperator { //this is not a real operator..
  // Options that appy to all operators:
  printBoth;      //[t] default = false
  onlyDiagonal;   //[t] default = false
  radialIntegral; //[b] default = false
  units;          //[t] default = au
}
```

- printBoth: Print <a|h|b> and <b|h|a> ? false by default. (For *some* operators, e.g., involving derivatives, this is a good test of numerical error. For most operators, values will be trivially the same; reduced matrix elements,

sign may be different.)

- onlyDiagonal: If true, will only print diagonal MEs <a|h|a>
- radialIntegral: calculate radial integral, or reduced matrix elements (difference depends on definition of operator in the code)
- units: can only be 'au' or 'MHz'. MHz only makes sense for some operators (e.g., hfs), and is just for convenience

Available operators:

```
MatrixElements::E1 { //Electric dipole operator:
  gauge; //[t] lform, vform. default = lform
}
```

```
MatrixElements::r { //scalar r
  power; //[r] default = 1. Will calc <|r^n|>.
}
```

```
MatrixElements::pnc {// spin-independent (Qw) PNC operator.
  // Output given in units of i(-Q/N)e-11
  c; //[r] half-density radius. By default, uses rrms from Z,A [see nucleus]
  t; //[r] skin thickness. default = 2.3
}
```

```
MatrixElements::hfs { // Magnetic dipole hyperfine structure constant A
  mu;     //[r] Nuc. mag. moment. Will be looked up by default
  I;      //[r] Nuc. spin. Will be looked up by default
  rrms;   //[r] Nuc. rms radius. Will be looked up by default
  F(r);   //[t] Bohr-Weisskopf function. ball, shell, pointlike, VolotkaBW,
doublyOddBW
  printF; //[b] default = false. Will write F(r)/r^2 to text file
  // -----
  // the following are only used for "VolotkaBW/doublyOddBW"
  // both are optional (will be deduced otherwise)
  parity; //[i] parity of unpaired valence nucleon (+1/-1)
  gl;     //[i] =1 for valence proton, =0 for valence neutron
  // -----
  // The following are only read in if F(r) = doublyOddBW,
  // but are _required_ in that case (current values are for 212-Fr)
  mu1 = 4.00; //see paper for explanation
  I1 = 4.5;
  l1 = 5.;
  gl1 = 1;
  I2 = 0.5;
  l2 = 1.;
}
```

Modules:

```
Module::Tests { // tests of numerical errors:
  orthonormal;     //[b] Prints worst <a|b>'s. default = true
  orthonormal_all; //[b] Print all <a|b>'s. default = false
  Hamiltonian;     //[b] check eigenvalues of Hamiltonian. default = false
  boundaries;      //[b] check f(rmax)/fmax. default = false
}
```

```
Module::BohrWeisskopf { //Calculates BW effect for Ball/Single-particle
  // Takes same input at MatrixElements::hfs*
  // *Except for F(r), since it runs for each F(r)
}
```

```
Module::WriteOrbitals { //writes orbitals to textfile:
    label = outputLabel; //[t] Optional. blank by default
}
```

Writes the core + valence orbitals (+ the total electron density) to a file, in GNUplot friendly format.
The (optional) label will be appended to the output file name. Plot file using GNUPLOT. For example, try this:

- *plot for [i=2:20] "file.txt" u 1:i every :::0::0 w l t columnheader(i)*
- *plot for [i=2:20] "file.txt" u 1:i every :::1::1 w l t columnheader(i)*
- *plot "file.txt" u 1:2 every :::2::2 w l t "Core Density"*

```
Module::FitParametric {
  method = Green;     //[t] Green, Tietz
  statesToFit = core; //[t] core, valence, both
  fitWorst;           //[b] false (default), true;
}
```

Performs a 2D fit to determine the best-fit values for the given two-parameter parametric potential (Green, or Tietz potentials), returns H/g d/t parameters for the best-fit. Does fit to Hartree-Fock energies. Will either do for core or valence states, or both (works best for one or the other). fitWorst: if true, will optimise fit for the worst state. If false, uses least squares for the fit. False is default

```
Module::pnc {
    //Calculates pnc amplitude {na,ka}->{nb,kb}
    //(these states must exist as valence states in HF!).
    // Uses sum-over-states method (just using the HF states as a basis),
    // and the Solving-Equations method (without RPA for now).
    transition = na, ka, nb, ka; //[t] - required
}
```

```
Module::AtomicKernal {
    // Some typical inputs. All are required.
  Emin = 0.01; // in keV
  Emax = 4.0;
  Esteps = 25;
  qmin = 0.001; // in MeV
  qmax = 4.0;
  qsteps = 100;
  max_l_bound = 1; // l for bound states
  max_L = 2;       // L is multipolarity
```

```
  output_text = true;
  output_binary = true;
  label = test_new;
  use_plane_waves = false;
}
```

Calculates the "Atomic Kernal" (for scattering/ionisation) for each core
orbital, as a function of momentum transfer (q), and energy deposition (dE).
Writes result to human-readable (and gnuplot-friendly) file, and/or binary.

- For definitions/details, see:
  - B.M. Roberts, V.V. Flambaum Phys.Rev.D 100, 063017 (2019); arXiv:1904.07127.
  - B.M.Roberts, V.A.Dzuba, V.V.Flambaum, M.Pospelov, Y.V.Stadnik, Phys.Rev.D 93, 115037 (2016); arXiv:1604.04559.
- Note: need quite a dense grid [large number of points] for
  - a) highly oscillating J_L function at low r, and
  - b) to solve equation for high-energy continuum states.
- Sums over 'all' continuum angular momentum states (and multipolarities)
  - Maximum values for l are input parameters
    Binary output from this program is read in by dmeXSection program
    Note: tested only for neutral atoms (V^N potential).
    Also: tested mainly for high values of q

Other programs:

periodicTable

Command-line periodic table, with electron configurations and nuclear data

- Compiled using the Makefile (run *$make*, must habe 'make' installed)
- Alternatively, compile with command:
  $g++ src/Physics/AtomData.cpp src/Physics/NuclearData.cpp src/periodicTable.cpp -o periodicTable -I./src/
- No other dependencies

Gives info regarding particular element, including Z, default A, and electron configuration.
Takes input in one line from command line.

Usage: (examples)

- *$./periodicTable* Prints periodic table
- *$./periodicTable Cs* Info for Cs with default A
- *$./periodicTable Cs 137* Info for Cs-137
- *$./periodicTable Cs all* Info for all available Cs isotopes
- Note: numbers come from online database, and have some errors,
  so should be checked if needed.

Or, enter 'c' Data to print list of physics constants)

- *$./periodicTable c* Prints values for some handy physical constants

Note: ground-state electron configurations are "guessed", and can sometimes be incorrect.

Nuclear radius data mostly comes from:

- I. Angeli and K. P. Marinova, At. Data Nucl. Data Tables 99, 69 (2013). https://doi.org/10.1016/j.adt.2011.12.006

Units:

- r_rms: root-mean-square radius, in fm.
- c: half-density radius (assuming Fermi nuclear distro)
- mu: magnetic moment (in nuclear magnetons)

dmeXSection

- Calculates the cross-section and event rates for ionisation of atoms by scattering of DM particle.
- Takes in the output of "atomicKernal" (see README_input for details)
- Also calculates "observable" event rates, accounting for detector thresholds and resolutions. For now, just for DAMA detector. Will add for XENON
- For definitions/details, see:
  - B.M. Roberts, V.V. Flambaum Phys.Rev.D 100, 063017 (2019); arXiv:1904.07127.
  - B.M.Roberts, V.A.Dzuba, V.V.Flambaum, M.Pospelov, Y.V.Stadnik, Phys.Rev.D 93, 115037 (2016); arXiv:1604.04559.

wigner

- Small routine to calculate 3,6,9-j symbols, and Clebsch Gordan coeficients
- Either give input via command line directly (quote marks required)
  - e.g., *./wigner '<0.5 -0.5, 0.5 0.5| 1 0>'*
  - or e.g., *./wigner '(0.5 1 0.5, -0.5 0 0.5)'* etc.
- Or, give an input file, that contains any number of symbols, all on new line
  - e.g., *./wigner -f myInputFile.in*
  - nb: the '-f' flag can be dropped in the '.in' file extension is used
  - Do not use quote marks in input file. Lines marked '!' or '#' are comments
- 3j symbols must start with '('; 6,9j with '{', and CG with '<' (this is how code knows which symbol to calculate).
- but, each number can be separated by any symbol (space, comma etc.)