Libint Notes

The Author

February 11, 2017

```
1 cd Software/kg_libint/build/tests/hartree-fock/
```

1 Libint Programmer's Manual

Objective : Figure out how compute works.
Libint library contains functions to compute many-body integrals over Gaussian basis functions which appear in electronic and molecular structure theories. Two-body Coulomb (electron repulsion) integrals.

2 Notes on Basis sets

Gaussian functions come in shells, which are sets of functions with same origin and radial dependence and only differing in their angular dependence. s-type shell (angular momentum l=0) has 1 function, p-type shell has 3, and so on. There are 2 kinds of shells: Cartesian and solid harmonic shells (the differences between the two only become apparent for l=2, e.g. there are 6 Cartesian d functions (xx, xy, xz, yy, yz, zz) but there are only 5 solid harmonic d Gaussians (think 2l+1 possible values of m _l). For your purposes this is irrelevant, just treat shells as groups of functions. So a basis can be viewed as a set of individual Gaussian functions (this is useful when thinking of matrix representation of operators) or as a set of shells (this is useful when computing integrals since taking advantage of the shell structure is essential to make the integral engine work efficiently). Consider water molecule in STO-3G basis. This is a so-called minimal basis: each atomic orbital in an atom is represented by a single Gaussian basis function. For the O atom this means there will be 1 function representing 1s orbital, 1 function representing 2s, and 3 functions representing 2p. We describe the structure of the STO-3G basis for O as 2s1p (2 s shells and 1 p shell), i.e. there are 3 shells on the O atom in this basis, and they correspond to 5 basis functions. For the H atom there is only 1 function and 1 s shell representing the 1s atomic orbital. For H2O in STO-3G we thus have 5 shells total (3 from O, 1 from each H), and 7 basis functions. When computing the 1-e integrals we loop over all 25 = 5 x

5 possible doublets of shell indices (we are forgetting about symmetry for the moment?), compute each shell block and copy the data from the integral engine to the corresponding location in the matrix that will store the complete set of 1-e ints. it is key to understand how 2 and higher-dimensional arrays are stored in memory. Let?s say we are computing a shell doublet of type ¡p — O — d¿ where p and d are p and d shells. There are 15 elements in this block (assuming solid harmonic d shell, which is the convention for all except the oldest basis sets). After calling compute the engine?s result pointer points to the first of the 15 elements of the resulting ¡p—d¿ shell set. In which order are they stored? They are packed as follows: ¡p_0—d_0¿, ¡p_0—d_1¿, ¡p_0—d_2¿, ¡p_0—d_3¿, ¡p_0—d_4¿, ¡p_1—d_0¿, ¡p_1—d_2¿, ? ¡p_2—d_4¿, where p_1 is the second function in the the p-shell, d_2 is the third function in the d-shell, etc. This is what we call a row-major packing of a 2-dimensional array (matrix). The idea is trivially extended to 3- and higher-dimensional arrays. Where should ¡p_0—d_0¿ be copied? It?s the first element in the shell doublet, but it needs to be copied to the matrix at the row and column combination that correspond to the start of each shell. The first basis function for a given shell (or the ?address" of the shell in the basis) of course depends on the basis composition (i.e., on the shells preceeding this one). Computing these shell addresses is easy but mundane. To simplify these computations you should use the BasisSet basis set class provided by libint. BasisSet makes it easy to construct a basis using a set of Atoms and the name of the basis (libint2 includes a library of most popular basis sets). To construct an STO-3G basis for a water molecule you would just do: “‘ auto basis = BasisSet(?STO-3G?, water); // water is a vector of Atom objects “‘ BasisSet is nothing but a std::vector¡Shell¿ (in fact, see the definition of BasisSet class at https://github.com/evaleev/libint/blob/master/include/libint2/basis.h.in#L44 ? note that it is derived from a vector of Shells, which means it inherits all the methods of std::vector, like size(), which returns the size of the vector, i.e. the number of shells, etc.). But it also provides a number of other functions that are very useful, e.g. BasisSet::nbf() returns the total number of functions in the basis (see https://github.com/evaleev/libint/blob/master/include/lib and BasisSet::shell2bf() returns the map from shell index to shell address in the basis (see https://github.com/evaleev/libint/blob/master/include/libint2/basis.h.in#L146). Note that Eigen::Map can be used to simplify copying computed shell doublets to the result Matrix, but since Eigen only officially supports matrices you will have to learn how to copy the data yourself. See e.g. https://github.com/evaleev/libint/blob/master/tests/hartree-fock/hartree-fock.cc#L632