*Self-Consistent Field Method - A NWChem module for evaluation*

PNNL has completed and delivered the first of two NWChem modules. The Self-Consistent Field Method (SCF) is often the central and most time consuming computation in ab initio quantum chemistry methods. It is used to solve the electronic Schrodinger Equation,

SCF assumes that each particle of the system is subjected to the mean field created by all other particles resulting in a non-linear system solvable by an iterative, fixed-point algorithm.

By expressing the molecular system’s one electron orbitals as the dot product of the system’s eigenvectors and a set of Gaussian basis functions

, [1]

the solution to the Schrodinger Equation reduces to the following self-consistent eigenvalue problem

[2]

[3]

[4]

where ***F*** is the Fock matrix, ***C*** are the eigenvectors of the system, ***ε*** are the eigenvalues, ***S*** is the electron force overlap matrix, ***D*** is the system density matrix, ***h*** are the one-electron forces, and the terms in the square brackets in Equation 2 are the two-electron Coulomb forces and the two-electron Exchange forces, respectively.

Figure 1 depicts the method’s control flow. The upper two leftmost modules initialize the ***D*** and ***C*** matrices allowing the first iteration to compute Equations 2 and 3. The *Construct Fock Matrix, Compute Orbitals,* and *Compute Density Matrix* modules compute Equations 2, 3, and 4, respectively. The *Damp Density Matrix* module scales the density matrix and finds the greatest changed value between the current and previous density matrix. If the absolute value of the change is less than a threshold value, the method terminates; otherwise, a new iteration is started. The method terminates after 30 iterations if convergence is not reached. Figure 2 gives the modules’ names, and array inputs and outputs.

The modules comprise rectangular, nested for loops of the form

**oneel**

for (i = 0; i < nbfn; i++) {  
for (j = 0; j < nbfn; j++) {  
 g\_fock[i][j] = (g\_schwarz[i][j] > tol) ? h(i,j) : 0.0;  
} }

**makden**

for (i = 0; i < nbfn; i++) {  
for (j = 0; j < nbfn; j++) {  
 double p = 0.0;  
 for (k = 0; k < nocc; k++) p += g\_orbs[i][k] \* g\_orbs[j][k];  
 g\_work[i][j] = 2.0 \* p;  
} }

**dendif**

for (i = 0; i < nbfn; i++) {  
for (j = 0; j < nbfn; j++) {  
 double xdiff = fabs(g\_dens[i][j] – g\_work[i][j]);  
 if (xdiff > denmax) denmax = xdiff;  
} }

where *nbfn* is the number of basis functions (= 15 \* number of atoms). The loops are embarrassingly parallel, but the inclusion of reduction operations such as sum and max require concurrent atomic updates.

The most computationally intensive routine is **twoel** that computes the two electron forces,

**twoel**

for (l = 0; l < nbfn; l++) {  
for (k = 0; k < nbfn; k++) {  
for (j = 0; j < nbfn; j++) {  
for (i = 0; i < nbfn; i++) {  
  
 if (g\_schwarz[i][j] \* schwmax ) < tol2e) {icut1 ++; continue;}  
 if (g\_schwarz[i][j] \* g\_schwarz[k][l]) < tol2e) {icut2 ++; continue;}  
  
 icut3 ++;  
 double gg = g(i, j, k, l);  
 g\_fock[i][j] += ( gg + g\_dens[k][l]);  
 g\_fock[i][k] -= (0.5 \* gg \* g\_dens[j][l]);  
  
} } } }

The time for all the other routines is much smaller than the time for twoel. The guards in the innermost loop impose a cutoff limit and reduce significantly the number of updates computed. The guards can be hoisted to reduce loop overhead as explained in the optimization file that accompanies the code release. We note that the ***h*** values (one electron forces) and ***g*** values (two electron forces) used in computing the Fock matrix are constants, so they could be precomputed, saved, and reused. While ***h*** is *nbfn*2, ***g*** is *nbfn*4 making it impractical to save ***g*** for even small molecular systems. As explained in the optimization file, symmetries in ***g*** can be exploited to minimize storage and computation requirements.

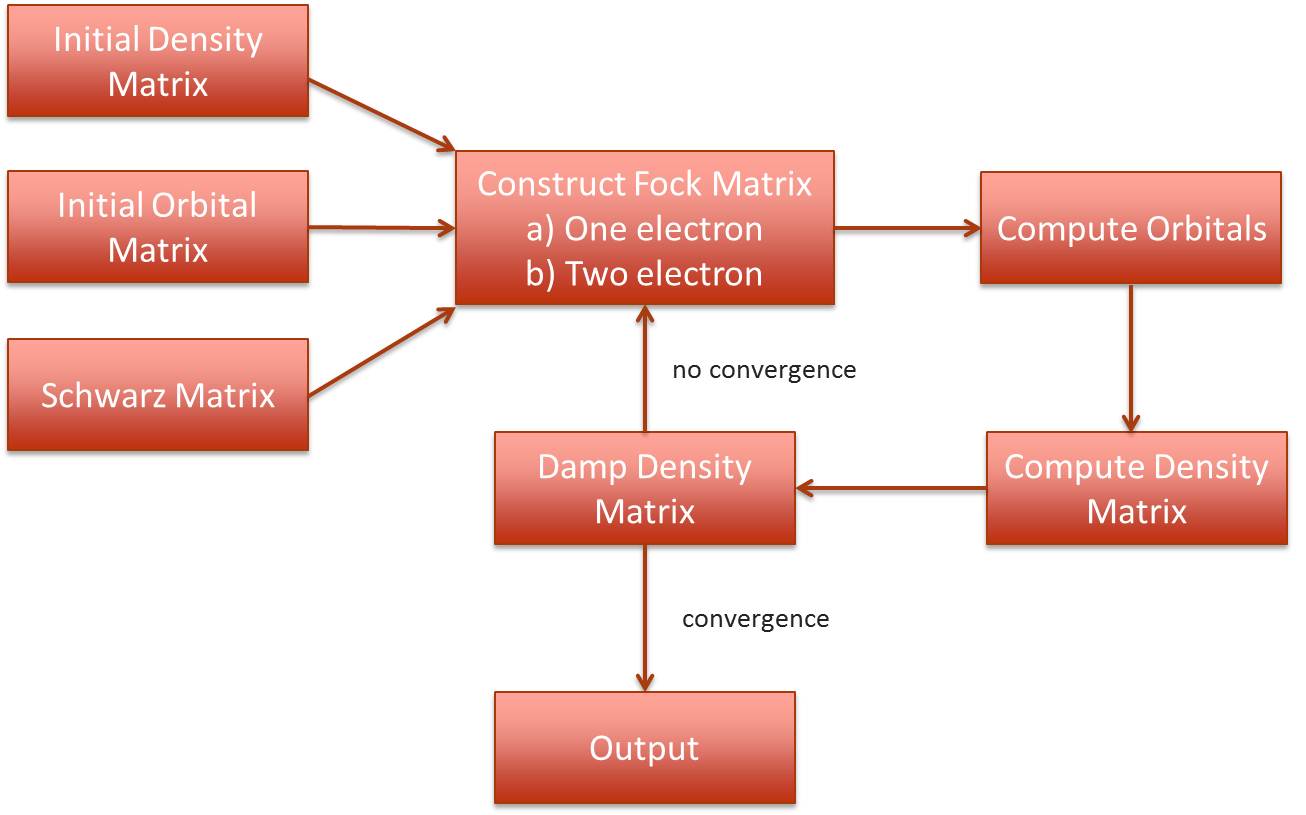


Figure 1 – SCF control flow

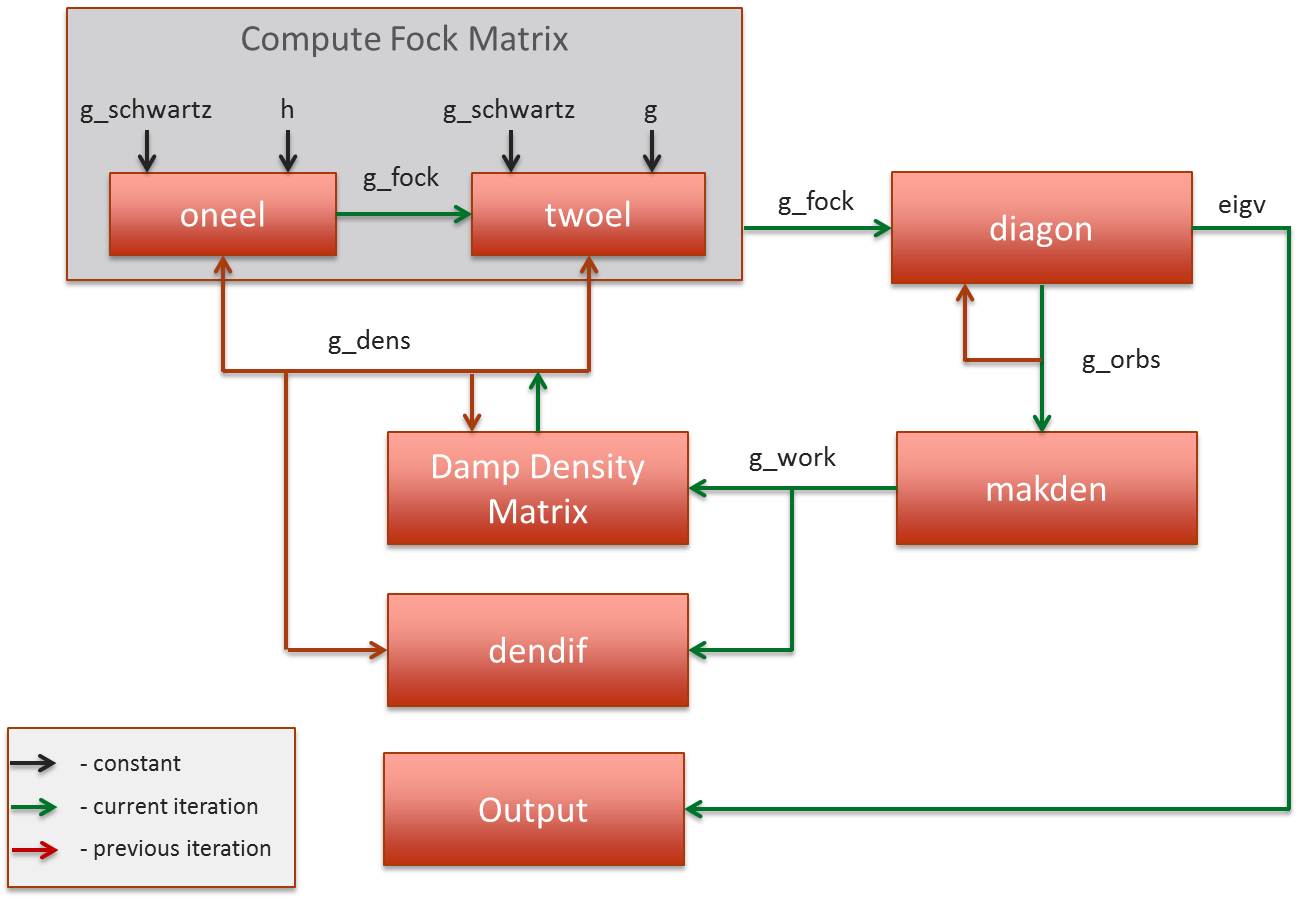


Figure 2 – SCF modules, inputs, and outputs