Hartree-Fock Program Project

C. David Sherrill
School of Chemistry and Biochemistry
Georgia Institute of Technology

Created on 3 May 2002
Based on notes by Dr. Yukio Yamaguchi (University of Georgia)

**Goals:** Write a computer program to perform a closed-shell restricted Hartree-Fock computation, given nuclear repulsion energy and one-electron integrals as input through the PSI3 program package.

**Procedure:** Here we will briefly outline the primary computational steps. In subsequent sections, we will discuss strategies for implementing these steps.

1. Reading in data
   (a) Read in the nuclear repulsion energy \((E_{\text{nuc}})\).
   (b) Read in the overlap integrals \((S)\).
   (c) Read in the kinetic energy integrals \((T)\).
   (d) Read in the potential energy integrals \((V)\).
   (e) Form the one-electron integrals \((H)\), via \(H_{\mu\nu} = T_{\mu\nu} + V_{\mu\nu}\).
   (f) Read in the two-electron integrals, \(\langle \mu\nu|\rho\sigma \rangle\).

2. Construct the orthogonalizing matrix \(S^{-1/2}\)
   (a) Diagonalize the \(S\) matrix,
   \[
   U^\dagger S U = \Lambda. \tag{1}
   \]
   (b) Form the \(S^{-1/2}\) matrix,
   \[
   S^{-1/2} = U \Lambda^{-1/2} U^\dagger. \tag{2}
   \]

3. Construct an initial (guess) density matrix
   (a) Form the “core” Fock matrix in the orthogonalized basis via
   \[
   F_0' = (S^{-1/2})^\dagger H S^{-1/2}. \tag{3}
   \]
(b) Diagonalize the initial Fock matrix using a standard eigenvalue routine such as the DGEMM routine in the BLAS library.

\[ C_0^r F_0^r C_0^r = \epsilon. \quad (4) \]

(c) Form the initial SCF eigenvector matrix in the original basis

\[ C_0 = S^{-1/2} C'_0. \quad (5) \]

(d) Form the initial density matrix, \( D \)

\[ D_{\mu\nu} = \sum_i \epsilon_i C_{\mu i} C_{\nu i}. \quad (6) \]

4. Perform the SCF iterations

(a) Form the new Fock matrix, \( F \)

\[ F_{\mu\nu} = H_{\mu\nu} + \sum_{\rho\sigma} D_{\rho\sigma} \{2[\mu\nu|\rho\sigma] - [\mu\rho|\nu\sigma]\}. \quad (7) \]

(b) Calculate the electronic energy

\[ E = \sum_{\mu\nu} \sum_{\mu'\nu'} D_{\mu\nu} (H_{\mu\nu} + F_{\mu\nu}) + E_{\text{nuc}}. \quad (8) \]

(c) Transform the Fock matrix to the orthonormal basis

\[ F' = (S^{-1/2})^T F S^{-1/2}. \quad (9) \]

(d) Diagonalize the Fock matrix

\[ C'^r F' C' = \epsilon. \quad (10) \]

(e) Construct the new SCF eigenvector matrix

\[ C = S^{-1/2} C' \quad (11) \]

(f) Form the new density matrix

\[ D_{\mu\nu} = \sum_i \epsilon_i C_{\mu i} C_{\nu i}. \quad (12) \]
(g) Test for convergence of density matrix and energy

\[ D_{\text{rms}} = \left[ \sum_{\mu \nu} \left( D_{\mu \nu}^n - D_{\mu \nu}^{n-1} \right)^2 \right]^{1/2} < \delta_D \quad (13) \]

\[ \Delta E = E^n - E^{n-1} < \delta_E \quad (14) \]

If not converged, do another iteration.

**Reading data from PSI3:** Here we will briefly describe interfacing to the PSI3 package. Additional information may be found in the PSI3 programmer’s manual. We will read the integrals using the `libpsio` and `libiw` libraries. It may also be useful to read relevant information from the “checkpoint” library, `libchkpt`, and other routines from `libciomr` may be helpful.

1. Consult the PSI programmer’s manual for info on how to set up a skeleton PSI program, and what initialization calls to make. Use this as your starting point. Then add the code below.

2. The S, T, and V integrals are written to disk as lower triangles of the full matrix, since \( S_{\mu \nu} = S_{\nu \mu} \), etc. For an \( nxn \) matrix, there are \((n(n + 1))/2\) elements in the lower triangle. Allocate an array of double’s this large and read in S via

\[
\text{errcod} = \text{iwl_rdone}(\text{PSIF} \_\text{OEI}, \text{PSIF} \_\text{SO} \_\text{S}, \text{ints}, \text{ntri}, 0, 0, \text{outfile});
\]

where `ints` is the array to store the integrals, `ntri` is the number of elements in the lower triangle, and `outfile` is the usual PSI output file pointer. See the PSI3 library documentation for `libiw` for more details on this routine. After reading in the S integrals, use similar operations to read T and V; the only thing that has to change is the label, which becomes `PSIF_SO_T` and `PSIF_SO_V`.

3. Read the two-electron integrals from disk. These are packed in an analogous way to the lower triangle of S, except that there are now four indices instead of two. PSI3 uses the fact that the integrals are eightfold symmetric, with \([\mu \nu | \rho \sigma] = [\nu \mu | \rho \sigma] = [\mu \nu | \sigma \rho] = [\rho \sigma | \mu \nu] = [\rho \sigma | \nu \mu] = [\sigma | \rho | \nu | \mu]. This allows us to only store the integrals such that \( \mu \geq \nu, \rho \geq \sigma, \) and \( \mu \nu \geq \rho \sigma \), where a compound index \( \mu \nu \) is computed as \( \mu \nu = (\mu (\mu + 1))/2 + \nu \) and \( \rho \sigma \) is defined similarly. A total index for the entire integral \([\mu \nu | \rho \sigma] (\mu \geq \nu, \rho \geq \sigma, \mu \nu \geq \rho \sigma) \) can be defined similarly as \( \mu \nu \rho \sigma = (\mu (\mu + 1))/2 + \rho \sigma \) using C numbering (starting from 0). The integrals may be read by allocating an appropriately sized buffer and calling

\[
\text{iwl_rdtwo}(\text{PSIF} \_\text{SO} \_\text{TEI}, \text{buffer}, \text{ioff}, \text{nso}, 0, 0, 0, \text{outfile});
\]
where \texttt{nso} is the number of symmetry-adapted orbitals (should be the number of atomic orbitals here, since you need to run in \textit{C}$_1$ symmetry unless you symmetry-adapt your code!) and \texttt{ioff} is a precomputed offset array such that

\[
\text{ioff}[p] = \frac{p(p+1)}{2}
\]

To read more about the permutational symmetry of integrals, the \texttt{ioff} array, and related subjects, consult the notes “Permutational Symmetries of Integrals” by David Sherrill.

4. Read \texttt{Enuc} using the checkpoint library \texttt{libchkpt}. To do this, \texttt{#include <libchkpt/chkpt.h>} and initialize the checkpoint library using \texttt{chkpt_init()}. Getting the nuclear repulsion energy is as easy as

\[
\text{Enuc} = \text{chkpt_rd_enuc}();
\]

Close the checkpoint file by calling \texttt{chkpt_close()}. 

5. The matrix diagonalization routine \texttt{DGEEV} and the matrix-matrix multiply routine \texttt{DGEMM} can be conveniently called using the C wrappers \texttt{C_DSYEV()} and \texttt{C_DGEMM()}. When using BLAS or LAPACK calls such as these, the subroutines assume that the matrix is available as one contiguous chunk of memory. This can be assured using the PSI matrix allocation routine \texttt{block_matrix()} (the routine \texttt{free_block()} frees up block matrices).
Makefile: To help you with linking to PSI and the math libraries, here is a sample Makefile. You will need to edit to point to your installed copy of PSI and your math and Fortran libraries (the long paths to the Fortran libraries are necessary because the lapack and blas libraries are Fortran libraries calling base-level Fortran functions).

SRC = hf.c
PSI = /theoryfs/ds/home/sherrill/psi3-bin
PSI_LIB = $(PSI)/lib
PSI_INCLUDE = $(PSI)/include
INCL =
OBJ = $(SRC:.c=.o)
EXE = hf
CC = gcc
CFLAGS = -ansi -g -I$(PSI_INCLUDE)
LDFLAGS = -o $(EXE)
LIBS = $(PSILIBS) \\
- L/usr/lib/gcc/x86_64-redhat-linux/4.0.2 \\
- L/usr/lib/gcc/x86_64-redhat-linux/4.0.2/../../../../../lib64 \\
- L/usr/lib/gcc/x86_64-redhat-linux/4.0.2/../../../../../lib64 \\
- L/usr/lib//lib64 -lgfortranbegin -lgfortran -lm -lgcc_s \\
- llblas -llapack

%.o: %.c
    $(CC) -c $(CFLAGS) $*.c

$(EXE): $(OBJ)
    $(CC) $(OBJ) $(LDFLAGS) $(LIBS)

$(OBJ): $(INCL)

clean:
    rm -f $(OBJ) $(EXE) core

NOTE: As in any makefile, the rule lines (e.g., $(CC) -c $(CFLAGS) $*.c) MUST begin with a tab character, not spaces.

Additional Information: See chapter 3 (sections 3.4.4 - 3.4.6) of Modern Quantum Chemistry, 1st Ed., Revised, A. Szabo and N. S. Ostlund (McGraw-Hill, New York, 1989).

Acknowledgment: This programming exercise comes from Dr. Yukio Yamaguchi, University of Georgia.