HF Final Project

Austin M. Wallace

2023-04-24

Hartree-Fock Overview

The objective of Hartree-Fock (HF) is to solve the Schrodinger Equation through the usage of optimizing coefficients for a linear combination of atomic orbitals (LCAO) to create molecular orbitals that minimize the electronic energy. Because this procedure depends on the electronic energy, the nuclear energy is not added until the very end of the self-consistent field procedure is complete. Within this procedure, orbitals can be optimized by minimizing the electronic energy due to the varitaional theorem.

Effectively, this process relies on nuclear coordinates, charge, multiplicity, and a basis set – the present work uses atom-centered Gaussian functions to describe atomic orbitals. From here, the overalp of atomic orbitals is computed to forumlate S along with compiting kinetic (T) and potential (V) energy intregals at the beginning. Next, the Core Hamiltonian is formed $H_{\mu\nu} = T_{\mu\nu} + V_{\mu\nu}$. Before constructing the core Fock matrix (Equation 1) for a naive initial density matrix guess (Equation 3), one must orthogonalize $S^{-1/2}$ through diagonalizing S and transforming to $S^{-1/2}$.

$$F_0' = (S^{-1/2})^\dagger H S^{-1/2} \tag{1}$$

Through diagonlizing the core Fock matrix with the orthogonalized basis, one can construct C_0.

$$C_0 = S^{-1/2} C_0' \tag{2}$$

Using these sorted eigenvectors, we compute our density guess.

$$D_{\mu\nu} = \sum_i^{N/2} C_{\mu i} C_{\nu i} \tag{3}$$

With these inital values, the SCF procedure may begin, where we iteratively compute a new Fock matrix (Equation 4), compute electronic energy (Equation 5), transform Fock matrix to the orthonormal basis (Equation 6), diagonlaize F, (Equation 7) update C (Equation 8), and update the density (Equation 9) until converging the energy.

$$F_{\mu\nu} = H_{\mu\nu} + \sum_{\rho\sigma}^{AO} D_{\rho\sigma} 2[\mu\nu|\rho\sigma] - [\mu\rho|\nu\sigma] \tag{4}$$

$$E = \sum_{\mu\nu}^{AO} D_{\mu\nu}(H_{\mu\nu} + F_{\mu\nu}) \tag{5}$$

$$F' = (S^{-1/2})^\dagger F S^{-1/2} \tag{6}$$

$$C'^\dagger F' C' = \epsilon \tag{7}$$

$$C = S^{-1/2} C' \tag{8}$$

$$D_{\mu\nu} = \sum_i^{N/2} C_{\mu i} C_{\nu i} \tag{9}$$

At the end of each iteration, convergence is checked by computing the energy and comparing with the previous iteration and comparing with a user defined threshold to break the iterative process. Once below the threshold, the electronic energy is added to the nuclear energy to get the HF energy.

Specific Implementation Details

I have implemented the abstract described HF procedure above in C++ through the usage of Eigen3 for performing linear algebra operations with Lapack under the hood. C++ was selected due to having strong support with OpenMP and MPI for parallelization and to improve fundamental knowledege for aspring to become a Psi4 developer.

The project allowed me to learn how to create my own cmake files for building the library across OSX and Ubuntu with ease using the pitchfork convention. Using CMakeLists.txt took some initial learning but provides a seamless path for finding libraries and repeatedly building the project during development. Additionally, it makes it easier for others to know what is needed for getting the project to run on their own machine.

The initial test case for getting the library working follows the "Coding Strategy #1" from canvas; however, the T, V, S, and ERI sections were broken into separate files for a simplier parsing function to generate Eigen::MatrixXd objects. The Eigen library provided an easy option for solving for eigenvalues and eigenvectors through passing your matrix to the SelfAdjointEigenSolver object initialization. The eigenvectors and eigenvalues already returned sorted and have methods to reverse the sorting if needed.

Code

The project is available on GitHub, where I plan to continue to implement parallelization towards a distributed HF code.

The most important files are `hf.cpp`, `input.cpp`, and `helper.cpp` which are displayed below.

```cpp
#include "helper.hpp"
#include "input.hpp"
#include "omp.h"
#include "stdio.h"
#include <Eigen/Dense>
#include <ctime>
#include <fstream>
#include <iostream>
#include <string>
#include <vector>

using namespace std;

void serial() {
  // Specify Data Path
  /* std::string dataPath = "data/t1"; */
  std::string dataPath = "data/t0";
  double t1 = 1e-8, t2 = 1e-8;

  //

  // Make pointers to store input data
  int num_atoms;
  double E = 0, e_nuc = 0;
  std::vector<int> *elements = nullptr;
  std::vector<double> *eri = nullptr;

```

```
28  Eigen::MatrixXd *coords = nullptr;
29  Eigen::MatrixXd *T = nullptr;
30  Eigen::MatrixXd *V = nullptr;
31  Eigen::MatrixXd *e1 = nullptr;
32  Eigen::MatrixXd *S = nullptr;
33
34  // Read Data
35  input::gatherData(dataPath, num_atoms, &elements, &eri, &coords, &T, &V, &e1,
36                    &S, &e_nuc);
37  cout << "e_nuc: " << e_nuc << endl;
38
39  // Starting HF Code
40
41  // Allocate Memory for Matrices
42  Eigen::MatrixXd *H = nullptr;
43  Eigen::MatrixXd *S_12 = nullptr;
44  Eigen::MatrixXd *F = nullptr;
45  Eigen::MatrixXd *C = nullptr;
46  Eigen::MatrixXd *C_0_prime = nullptr;
47  Eigen::MatrixXd *D = nullptr;
48
49  // Allocate memory for energy and electron count
50  int num_electrons;
51  /* int eriSize; */
52
53  // Set Number of Electrons for a Neutral Molecule
54  helper::getNumberOfElectrons(num_atoms, elements, &num_electrons);
55
56  H = new Eigen::MatrixXd(T->rows(), T->cols());
57  *H = *T + *V;
58  cout << endl << "H Matrix: " << endl << endl << *H << endl;
59
60  // Orthogonalization of S
61  cout << endl << "S Matrix: " << endl << endl << *S << endl;
62  S_12 = new Eigen::MatrixXd(S->rows(), S->cols());
63  helper::orthoS(S, S_12);
64  cout << endl << "S_12 Matrix: " << endl << endl << *S_12 << endl;
65
66  // Build initial Fock Matrix
67  F = new Eigen::MatrixXd(H->rows(), H->cols());
68  *F = (*S_12).transpose() * *H * (*S_12);
69  cout << endl << "F Matrix: " << endl << endl << *F << endl;
70
71  C_0_prime = new Eigen::MatrixXd(H->rows(), H->cols());
72  helper::getC_0_prime(F, C_0_prime);
73  cout << endl << "C_0_prime Matrix: " << endl << endl << *C_0_prime << endl;
74
75  C = new Eigen::MatrixXd(H->rows(), H->cols());
76  *C = (*S_12) * (*C_0_prime);
77  cout << endl << "C Matrix: " << endl << endl << *C << endl;
78
79  D = new Eigen::MatrixXd(H->rows(), H->cols());
80  helper::updateDensityMatrix(C, D, num_electrons);
81  cout << endl << "D Matrix: " << endl << endl << *D << endl;
82
83  helper::SCF(eri, S_12, H, F, C, D, C_0_prime, num_electrons, &E, e_nuc, t1,
84              t2);
85  E += e_nuc;
86  cout.precision(15);
```

```
  cout << endl << "Final HF Energy: " << E << endl;

  // Final HF Energy:    -74.9659010585405
  // Total Energy =      -74.9659900701199433

  // Free Allocations
  free(elements);
  free(coords);
  free(T);
  free(V);
  free(e1);
  free(S);
  free(eri);
  printf("\nFreed Memory\n");
}

int main() {
  omp_set_num_threads(1);
  Eigen::setNbThreads(1);
  time_t start, end;
  double serial_t;
  start = clock();
  serial();
  end = clock();
  serial_t = (double)(end - start);
  cout << "Serial Time: " << (serial_t / CLOCKS_PER_SEC) << endl;
  double omp_t;
  int num_threads = 2;
  omp_set_num_threads(num_threads);
  Eigen::setNbThreads(num_threads);
  start = clock();
  serial();
  end = clock();
  omp_t = (double)(end - start);
  cout << "Omp Time: " << (double) (omp_t / CLOCKS_PER_SEC) << endl;
  cout << "Omp Speedup: " << (double)(serial_t / omp_t)
       << endl;

  return 0;
}
```

```
#include "input.hpp"
#include "helper.hpp"
#include <Eigen/Dense>
#include <algorithm>
#include <fstream>
#include <iostream>
#include <sstream>
#include <string>
#include <vector>

using namespace std;

void input::readVector(std::string fn, std::vector<std::vector<double>> **arr) {
  std::ifstream file(fn);
  if (!file) {
    std::cout << "Could not open file: " << fn << std::endl;
    return;
  }
```

```
19    std::string line;
20    int count = 0;
21
22    while (getline(file, line)) {
23      count++;
24    }
25    file.clear();
26    file.seekg(0);
27
28    *arr = new std::vector<std::vector<double>>(count);
29    for (int i = 0; i < count; ++i) {
30      (*arr)->at(i) = std::vector<double>(count);
31    }
32
33    int i = 0, j;
34    while (getline(file, line)) {
35      j = 0;
36      std::stringstream ss(line);
37      std::vector<double> values;
38      double value;
39      while (ss >> value) {
40        (*arr)->at(i).at(j) = value;
41        /* std::cout << value << " "; */
42        j++;
43      }
44      /* std::cout << std::endl; */
45      i++;
46    }
47    file.close();
48    return;
49  }
50
51  void input::readVector(std::string fn, Eigen::MatrixXd **arr) {
52    std::ifstream file(fn);
53    if (!file) {
54      std::cout << "Could not open file: " << fn << std::endl;
55      return;
56    }
57    std::string line;
58    int count = 0;
59
60    while (getline(file, line)) {
61      count++;
62    }
63    file.clear();
64    file.seekg(0);
65
66    *arr = new Eigen::MatrixXd(count, count);
67
68    int i = 0, j;
69    while (getline(file, line)) {
70      j = 0;
71      std::stringstream ss(line);
72      std::vector<double> values;
73      double value;
74      while (ss >> value) {
75        /* std::cout << value << " " << i << " " << j << std::endl; */
76        (**arr)(i, j) = value;
77        j++;
```

```
78      }
79      /* std::cout << std::endl; */
80      i++;
81    }
82    file.close();
83    return;
84  }
85  void input::readVector(std::string fn, std::vector<double> **arr) {
86    // TODO: need to read ERI into Nx4 matrix and use IJKL
87    // indexing to build eri reduced matrix
88    std::ifstream file(fn);
89    if (!file) {
90      std::cout << "Could not open file: " << fn << std::endl;
91      return;
92    }
93    std::string line;
94    int count = 0;
95
96    while (getline(file, line)) {
97      count++;
98    }
99    file.clear();
100   file.seekg(0);
101
102   *arr = new std::vector<double>(count);
103
104   int i = 0;
105   while (getline(file, line)) {
106     std::stringstream ss(line);
107     ss >> (*arr)->at(i);
108     i++;
109   }
110   file.close();
111   return;
112 }
113
114 void input::readERI(std::string fn, std::vector<double> **arr, int n_basis) {
115   std::ifstream file(fn);
116   if (!file) {
117     std::cout << "Could not open file: " << fn << std::endl;
118     return;
119   }
120   std::string line;
121   int count = 0;
122
123   while (getline(file, line)) {
124     count++;
125   }
126   file.clear();
127   file.seekg(0);
128   int arrSize =
129       /* n_basis * (n_basis + 1) / 2  * (n_basis + 2) / 2 * (n_basis  + 3) / 2 ;
130        */
131       /* n_basis * n_basis * n_basis * n_basis / 8; */
132       /* helper::indexIJKL(n_basis, n_basis, n_basis, n_basis); */
133       helper::indexIJKL(n_basis - 1, n_basis - 1, n_basis - 1, n_basis - 1) + 1;
134
135   cout << "arrSize: " << arrSize << endl;
136   cout << "count: " << count << endl;
```

```
137
138   *arr = new std::vector<double>(count);
139
140   int i, j, k, l;
141   double value;
142   int ijkl;
143   while (getline(file, line)) {
144     std::stringstream ss(line);
145     ss >> i >> j >> k >> l >> value;
146     ijkl = helper::indexIJKL(i, j, k, l);
147     /* std::cout << ijkl << " " << i << " " << j << " " << k << " " << l << " "
148      */
149     /*           << value << std::endl; */
150     (*arr)->at(ijkl) = value;
151   }
152   file.close();
153   return;
154 }
155
156 void input::gatherData(std::string dataPath, int &num_atoms,
157                        std::vector<int> **elements, std::vector<double> **eri,
158                        std::vector<std::vector<double>> **coords,
159                        std::vector<std::vector<double>> **T,
160                        std::vector<std::vector<double>> **V,
161                        std::vector<std::vector<double>> **e1,
162                        std::vector<std::vector<double>> **overlap
163
164 ) {
165   std::string geom = dataPath + "/geom.xyz";
166   std::string eriFN = dataPath + "/eri.dat";
167   std::string TFN = dataPath + "/T.dat";
168   std::string VFN = dataPath + "/V.dat";
169   std::string e1FN = dataPath + "/e1.dat";
170   std::string overlapFN = dataPath + "/overlap.dat";
171   // Gathering Geometry
172   input::readGeometry(geom, num_atoms, elements, coords);
173   std::cout << "Number of atoms: " << num_atoms << std::endl;
174   input::printElements(*elements);
175   input::printVector(*coords);
176
177   // Gathering T, V, e1, overlap
178   input::readVector(TFN, T);
179   /* input::printVector(*T); */
180   input::readVector(VFN, V);
181   /* input::printVector(*V); */
182   input::readVector(e1FN, e1);
183   /* input::printVector(*e1); */
184   input::readVector(overlapFN, overlap);
185   input::readVector(eriFN, eri);
186   /* printVector(*eri); */
187 }
188
189 void input::gatherData(std::string dataPath, int &num_atoms,
190                        std::vector<int> **elements, std::vector<double> **eri,
191                        Eigen::MatrixXd **coords, Eigen::MatrixXd **T,
192                        Eigen::MatrixXd **V, Eigen::MatrixXd **e1,
193                        Eigen::MatrixXd **overlap, double *enuc) {
194   std::string geom = dataPath + "/geom.xyz";
195   std::string eriFN = dataPath + "/eri.dat";
```

```
196   std::string TFN = dataPath + "/T.dat";
197   std::string VFN = dataPath + "/V.dat";
198   std::string e1FN = dataPath + "/e1.dat";
199   std::string overlapFN = dataPath + "/overlap.dat";
200   std::string enucFN = dataPath + "/enuc.dat";
201   // Gathering Geometry
202   input::readGeometry(geom, num_atoms, elements, coords);
203   /* std::cout << "Number of atoms: " << num_atoms << std::endl; */
204   /* input::printElements(*elements); */
205   /* input::printVector(*coords); */
206
207   // Gathering T, V, e1, overlap
208   input::readVector(TFN, T);
209   input::readVector(VFN, V);
210   input::readVector(e1FN, e1);
211   input::readVector(overlapFN, overlap);
212
213   // Gathering eri
214   /* input::readVector(eriFN, eri); */
215   int n_basis = (*T)->rows();
216   cout << "n_basis: " << n_basis << endl;
217   input::readERI(eriFN, eri, n_basis);
218   input::readNumber(enucFN, *enuc);
219 }
220
221 void input::numAtoms(std::string filename, int &num_atoms) {
222   std::ifstream file(filename);
223   if (!file) {
224     std::cout << "Could not open file " << filename << std::endl;
225     return;
226   }
227   file >> num_atoms;
228 }
229
230 void input::readGeometry(std::string filename, int &num_atoms,
231                          std::vector<int> **elements,
232                          Eigen::MatrixXd **coords) {
233   std::ifstream file(filename);
234   if (!file) {
235     std::cout << "Could not open file " << filename << std::endl;
236     return;
237   }
238
239   file >> num_atoms;
240   *elements = new std::vector<int>(num_atoms);
241   *coords = new Eigen::MatrixXd(num_atoms, 3);
242
243   std::string line;
244   std::getline(file, line); // read in the comment line
245
246   for (int i = 0; i < num_atoms; ++i) {
247     int el = -1;
248     double x, y, z;
249     file >> el >> x >> y >> z;
250     (*elements)->at(i) = el;
251     (**coords)(i, 0) = x;
252     (**coords)(i, 1) = y;
253     (**coords)(i, 2) = z;
254   }
```

```
255   file.close();
256   return;
257 }
258
259 void input::readGeometry(std::string filename, int &num_atoms,
260                          std::vector<int> **elements,
261                          std::vector<std::vector<double>> **coords) {
262   std::ifstream file(filename);
263   if (!file) {
264     std::cout << "Could not open file " << filename << std::endl;
265     return;
266   }
267
268   file >> num_atoms;
269   *elements = new std::vector<int>(num_atoms);
270   *coords = new std::vector<std::vector<double>>(num_atoms);
271   for (int i = 0; i < num_atoms; ++i) {
272     (*coords)->at(i) = std::vector<double>(3);
273   }
274
275   std::string line;
276   std::getline(file, line); // read in the comment line
277
278   for (int i = 0; i < num_atoms; ++i) {
279     int el = -1;
280     double x, y, z;
281     file >> el >> x >> y >> z;
282     (*elements)->at(i) = el;
283     (*coords)->at(i).at(0) = x;
284     (*coords)->at(i).at(1) = y;
285     (*coords)->at(i).at(2) = z;
286   }
287   file.close();
288   return;
289 }
290
291 void input::printVector(std::vector<std::vector<double>> *matrix) {
292   std::cout << std::endl;
293   for (int i = 0; u_int64_t(i) < matrix->size(); ++i) {
294     for (int j = 0; u_int64_t(j) < matrix->at(i).size(); ++j) {
295       std::cout.precision(12);
296       std::cout << matrix->at(i).at(j) << " ";
297     }
298     std::cout << std::endl;
299   }
300   std::cout << std::endl;
301 }
302
303 void input::printVector(std::vector<double> *matrix) {
304   std::cout << std::endl;
305   for (int i = 0; u_int64_t(i) < matrix->size(); ++i) {
306     std::cout.precision(12);
307     std::cout << matrix->at(i) << " ";
308   }
309   std::cout << std::endl;
310 }
311
312 void input::printElements(std::vector<int> *matrix) {
313   for (int i = 0; u_int64_t(i) < matrix->size(); ++i) {
```

```
    std::cout << matrix->at(i) << " ";
    std::cout << std::endl;
  }
}

void input::readNumber(std::string filename, double &number) {
  std::ifstream file(filename);
  if (!file) {
    std::cout << "Could not open file " << filename << std::endl;
    return;
  }
  file >> number;
  file.close();
}
```

```
#include "helper.hpp"
#include "stdio.h"
#include <Eigen/Dense>
#include <iostream>
#include <vector>

using namespace Eigen;
using namespace std;

void helper::orthoS(Eigen::MatrixXd *S, Eigen::MatrixXd *S12) {
  // Diagonalize S
  Eigen::SelfAdjointEigenSolver<Eigen::MatrixXd> es(*S);
  /* Eigen::MatrixXd D = es.eigenvalues().asDiagonal(); */
  Eigen::MatrixXd LAMBDA = es.eigenvalues().asDiagonal();
  Eigen::MatrixXd U = es.eigenvectors();
  // Invert D
  for (int i = 0; i < LAMBDA.rows(); i++) {
    LAMBDA(i, i) = 1 / sqrt(LAMBDA(i, i));
  }
  // Calculate X
  *S12 = U * LAMBDA * U.transpose();
}

void helper::initialFockMatrix(Eigen::MatrixXd *X, Eigen::MatrixXd *H,
                               Eigen::MatrixXd *F) {
  // Calculate F
  *F = (*X).transpose() * *H * (*X);
}

void helper::getC_0_prime(Eigen::MatrixXd *F, Eigen::MatrixXd *C) {
  // Diagonalize F
  SelfAdjointEigenSolver<MatrixXd> eigensolver(*F);
  if (eigensolver.info() != Success)
    abort(); // check for errors
  *C = eigensolver.eigenvectors();
}

void helper::computeEnergy(Eigen::MatrixXd *D, Eigen::MatrixXd *H,
                           Eigen::MatrixXd *F, double *E) {
  // Calculate E
  // TODO: fix this
  *E = 0;
  /* for (int i = 0; i < H->rows(); i++) { */
  /*   for (int j = 0; j < H->rows(); j++) { */
```

```
45    /*     *E += (*D)(i, j) * ((*H)(i, j) + (*F)(i, j)); */
46    /*   } */
47    /* } */
48    *E = (*D).cwiseProduct((*H) + (*F)).sum();
49    /* *E = (*D * ( (*H) + (*E))); */
50  }
51
52  void helper::getNumberOfElectrons(int num_atoms, std::vector<int> *elements,
53                                    int *num_electrons) {
54    // Calculate number of electrons
55    *num_electrons = 0;
56    for (int i = 0; i < num_atoms; i++) {
57      *num_electrons += elements->at(i);
58    }
59  }
60
61  int helper::indexIJKL(int i, int j, int k, int l) {
62    if (j > i){
63      std::swap(i, j);
64    }
65    if (l > k){
66      std::swap(k, l);
67    }
68    int ij = i * (i + 1) / 2 + j;
69    int kl = k * (k + 1) / 2 + l;
70    if (ij < kl){
71      std::swap(ij, kl);
72    }
73    int ijkl = ij * (ij + 1) / 2 + kl;
74    return ijkl;
75  }
76
77  void helper::updateDensityMatrix(Eigen::MatrixXd *C, Eigen::MatrixXd *D,
78                                   int num_electrons) {
79    // Calculate D
80      for (int i = 0; i < C->rows(); i++) {
81        for (int j = 0; j < C->rows(); j++) {
82          (*D)(i, j) = 0;
83          for (int k = 0; k < num_electrons / 2; k++) {
84            (*D)(i, j) += (*C)(i, k) * (*C)(j, k);
85          }
86        }
87      }
88  }
89
90  // TODO: finish this function
91  void helper::eriReducedCalc(std::vector<double> *eri,
92                              std::vector<double> *eriReduced) {
93    /* for (int i =0; i < eri->size(); i++){ */
94    /*     eriReduced->at(eri->at(i)); */
95    /* } */
96  }
97
98  void helper::updateFockMatrix(Eigen::MatrixXd *H, Eigen::MatrixXd *D,
99                                Eigen::MatrixXd *F, std::vector<double> *eri) {
100   // Update Fock Matrix
101   *F = *H;
102   for (int mu = 0; mu < H->rows(); mu++) {
103     for (int nu = 0; nu < H->cols(); nu++) {
```

```
      for (int rho = 0; rho < H->rows(); rho++) {
        for (int sig = 0; sig < H->cols(); sig++) {
          (*F)(mu, nu) += (*D)(rho, sig) *
                          (2 * eri->at(helper::indexIJKL(mu, nu, rho, sig)) -
                           eri->at(helper::indexIJKL(mu, rho, nu, sig)));
        }
      }
    }
  }
}

void helper::SCF(std::vector<double> *eri, Eigen::MatrixXd *S_12,
                 Eigen::MatrixXd *H, Eigen::MatrixXd *F, Eigen::MatrixXd *C,
                 Eigen::MatrixXd *D, Eigen::MatrixXd *C_0_prime,
                 int num_electrons, double *E, double e_nuc, double t1,
                 double t2) {
  // Calculate SCF

  bool converged = false;
  double E2 = 0;
  int iter = 0, max_iter = 100;
  while (!converged) {
    // Update Fock Matrix
    helper::updateFockMatrix(H, D, F, eri);
    /* cout << endl <<"F Matrix: " << endl << endl <<*F << endl; */
    /* cout << endl <<"E: " << endl << endl <<*E << endl; */
    helper::computeEnergy(D, H, F, E);
    *F = (*S_12).transpose() * *F * (*S_12);

    helper::getC_0_prime(F, C_0_prime);
    /* cout << endl <<"C_0_prime Matrix: " << endl <<endl << *C_0_prime << endl;
     */

    *C = (*S_12) * (*C_0_prime);
    /* cout << endl <<"C Matrix: " << endl <<endl << *C << endl; */
    helper::updateDensityMatrix(C, D, num_electrons);
    /* cout << endl <<"D Matrix: " << endl << endl <<*D << endl; */

    cout << "iter: " << iter << " Energy: " << *E << " Delta E: " << (*E - E2) << endl
         ↪ ;
    if (abs(*E - E2) < t1) {
      converged = true;
    } else if (iter > max_iter) {
      cout << "Max iterations reached" << endl;
      converged = true;
    } else {
      E2 = *E;
    }
    iter++;
    /* converged = true; */
  }
}
```