

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 variational 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 overlap of atomic orbitals is computed to formulate S along with computing kinetic (T) and potential (V) energy integrals 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 $S^{-1/2}$.

$$F'_0 = (S^{-1/2})^\dagger H S^{-1/2} \quad (1)$$

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

$$C_0 = S^{-1/2} C'_0 \quad (2)$$

Using these sorted eigenvectors, we compute our density guess.

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

With these initial 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), diagonalize F , (Equation 7) update C (Equation 8), and update the density (Equation 9) until converging the energy.

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

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

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

$$C'^\dagger F' C' = \epsilon \quad (7)$$

$$C = S^{-1/2} C' \quad (8)$$

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

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 knowledge for aspiring 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 simpler parsing function to generate Eigen::MatrixXd objects. For the larger system testing, the input data is generated with Psi4 through the Python3 API. 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. Also, the data generator for other systems is seen in the `main.py` file.

hf.cpp

```
1  #include "helper.hpp"
2  #include "input.hpp"
3  #include "omp.h"
4  #include "stdio.h"
5  #include <Eigen/Dense>
6  #include <ctime>
7  #include <fstream>
8  #include <iostream>
9  #include <string>
10 #include <vector>
11
12 using namespace std;
13
14 void HF_og() {
15     // Specify Data Path
16     /* std::string dataPath = "data/t1"; */
17     std::string dataPath = "data/t0";
18     double t1 = 1e-8, t2 = 1e-8;
19
20     // Make pointers to store input data
21     int num_atoms;
22     double E = 0, e_nuc = 0;
```

```

23  std::vector<int> *elements = nullptr;
24  std::vector<double> *eri = nullptr;
25
26  Eigen::MatrixXd *coords = nullptr;
27  Eigen::MatrixXd *T = nullptr;
28  Eigen::MatrixXd *V = nullptr;
29  Eigen::MatrixXd *e1 = nullptr;
30  Eigen::MatrixXd *S = nullptr;
31
32  // Read Data
33  input::gatherData(dataPath, num_atoms, &elements, &eri, &coords, &T, &V, &e1,
34                    &S, &e_nuc);
35  cout << "e_nuc: " << e_nuc << endl;
36
37  // Starting HF Code
38  // Allocate Memory for Matrices
39  Eigen::MatrixXd *H = nullptr;
40  Eigen::MatrixXd *S_12 = nullptr;
41  Eigen::MatrixXd *F = nullptr;
42  Eigen::MatrixXd *C = nullptr;
43  Eigen::MatrixXd *C_0_prime = nullptr;
44  Eigen::MatrixXd *D = nullptr;
45
46  // Allocate memory for energy and electron count
47  int num_electrons;
48
49  // Set Number of Electrons for a Neutral Molecule
50  helper::getNumberOfElectrons(num_atoms, elements, &num_electrons);
51
52  H = new Eigen::MatrixXd(T->rows(), T->cols());
53  *H = *T + *V;
54  cout << endl << "H Matrix: " << endl << endl << *H << endl;
55
56  // Orthogonalization of S
57  cout << endl << "S Matrix: " << endl << endl << *S << endl;
58  S_12 = new Eigen::MatrixXd(S->rows(), S->cols());
59  helper::orthoS(S, S_12);
60  cout << endl << "S_12 Matrix: " << endl << endl << *S_12 << endl;
61
62  // Build initial Fock Matrix
63  F = new Eigen::MatrixXd(H->rows(), H->cols());
64  *F = (*S_12).transpose() * *H * (*S_12);
65  cout << endl << "F Matrix: " << endl << endl << *F << endl;
66
67  C_0_prime = new Eigen::MatrixXd(H->rows(), H->cols());
68  helper::getC_0_prime(F, C_0_prime);
69  cout << endl << "C_0_prime Matrix: " << endl << endl << *C_0_prime << endl;
70
71  C = new Eigen::MatrixXd(H->rows(), H->cols());
72  *C = (*S_12) * (*C_0_prime);
73  cout << endl << "C Matrix: " << endl << endl << *C << endl;
74
75  D = new Eigen::MatrixXd(H->rows(), H->cols());
76  helper::updateDensityMatrix(C, D, num_electrons);
77  cout << endl << "D Matrix: " << endl << endl << *D << endl;
78
79  helper::SCF(eri, S_12, H, F, C, D, C_0_prime, num_electrons, &E, e_nuc, t1,
80             t2);
81  E += e_nuc;

```

```

82     cout.precision(15);
83     cout << endl << "Final HF Energy: " << E << endl;
84
85     // Free Allocations
86     free(elements);
87     free(coords);
88     free(T);
89     free(V);
90     free(e1);
91     free(S);
92     free(eri);
93     printf("\nFreed Memory\n");
94 }
95
96 void HF(int num_atoms, double E = 0, double e_nuc = 0,
97         std::vector<int> *elements = nullptr,
98         std::vector<double> *eri = nullptr, Eigen::MatrixX<double> *coords = nullptr,
99         Eigen::MatrixX<double> *T = nullptr, Eigen::MatrixX<double> *V = nullptr,
100         Eigen::MatrixX<double> *S = nullptr
101 ) {
102     // Specify Data Path
103     double t1 = 1e-8, t2 = 1e-8;
104
105     // Starting HF Code
106     // Allocate Memory for Matrices
107     Eigen::MatrixX<double> *H = nullptr;
108     Eigen::MatrixX<double> *S_12 = nullptr;
109     Eigen::MatrixX<double> *F = nullptr;
110     Eigen::MatrixX<double> *C = nullptr;
111     Eigen::MatrixX<double> *C_0_prime = nullptr;
112     Eigen::MatrixX<double> *D = nullptr;
113
114     // Allocate memory for energy and electron count
115     int num_electrons;
116     // Set Number of Electrons for a Neutral Molecule
117     helper::getNumberOfElectrons(num_atoms, elements, &num_electrons);
118
119     H = new Eigen::MatrixX<double>(T->rows(), T->cols());
120     *H = *T + *V;
121     /* cout << endl << "H Matrix: " << endl << endl << *H << endl; */
122
123     // Orthogonalization of S
124     /* cout << endl << "S Matrix: " << endl << endl << *S << endl; */
125     S_12 = new Eigen::MatrixX<double>(S->rows(), S->cols());
126     helper::orthoS(S, S_12);
127     /* cout << endl << "S_12 Matrix: " << endl << endl << *S_12 << endl; */
128
129     // Build initial Fock Matrix
130     F = new Eigen::MatrixX<double>(H->rows(), H->cols());
131     *F = (*S_12).transpose() * *H * (*S_12);
132     /* cout << endl << "F Matrix: " << endl << endl << *F << endl; */
133
134     C_0_prime = new Eigen::MatrixX<double>(H->rows(), H->cols());
135     helper::getC_0_prime(F, C_0_prime);
136     /* cout << endl << "C_0_prime Matrix: " << endl << endl << *C_0_prime << endl;
137     */
138
139     C = new Eigen::MatrixX<double>(H->rows(), H->cols());

```

```

141 *C = (*S_12) * (*C_0_prime);
142 /* cout << endl << "C Matrix: " << endl << endl << *C << endl; */
143
144 D = new Eigen::MatrixXd(H->rows(), H->cols());
145 helper::updateDensityMatrix(C, D, num_electrons);
146 /* cout << endl << "D Matrix: " << endl << endl << *D << endl; */
147
148 helper::SCF(eri, S_12, H, F, C, D, C_0_prime, num_electrons, &E, e_nuc, t1,
149             t2);
150 E += e_nuc;
151 cout.precision(15);
152 cout << endl << "Final HF Energy: " << E << endl;
153
154 // Free Allocations
155 free(elements);
156 free(coords);
157 free(T);
158 free(V);
159 free(S);
160 free(eri);
161 printf("\nFreed Memory\n");
162 }
163
164 void timings(std::string dataPath, int num_threads) {
165     int num_atoms;
166     double E = 0, e_nuc = 0;
167     time_t start, end;
168     double itime, ftime, exec_time;
169
170     std::vector<int> *elements = nullptr;
171     std::vector<double> *eri = nullptr;
172
173     Eigen::MatrixXd *coords = nullptr;
174     Eigen::MatrixXd *T = nullptr;
175     Eigen::MatrixXd *V = nullptr;
176     Eigen::MatrixXd *S = nullptr;
177     input::gatherData(dataPath, num_atoms, &elements, &eri, &coords, &T, &V, &S,
178                       &e_nuc);
179     omp_set_num_threads(num_threads);
180     Eigen::setNbThreads(num_threads);
181     double totTime;
182     omp_set_num_threads(num_threads);
183     Eigen::setNbThreads(num_threads);
184     start = clock();
185     itime = omp_get_wtime();
186     HF(num_atoms, E, e_nuc, elements, eri, coords, T, V, S);
187     ftime = omp_get_wtime();
188     end = clock();
189     totTime = (double)(end - start);
190     exec_time = ftime - itime;
191
192     cout << "Time (CPU) : " << (double)(totTime / CLOCKS_PER_SEC) << endl;
193     cout << "Time (USR) : " << exec_time << endl << endl;
194 }
195
196 void timings_parallel(std::string dataPath, int num_threads) {
197     int num_atoms;
198     double E = 0, e_nuc = 0;
199     time_t start, end;

```

```

200 double serial_t;
201 double itime, ftime, exec_time;
202
203 // Required code for which execution time needs to be computed
204 std::vector<int> *elements = nullptr;
205 std::vector<double> *eri = nullptr;
206 Eigen::MatrixX<double> *coords = nullptr;
207 Eigen::MatrixX<double> *T = nullptr;
208 Eigen::MatrixX<double> *V = nullptr;
209 Eigen::MatrixX<double> *S = nullptr;
210 input::gatherData(dataPath, num_atoms, &elements, &eri, &coords, &T, &V, &S,
211                  &e_nuc);
212 omp_set_num_threads(1);
213 Eigen::setNbThreads(1);
214 start = clock();
215 HF(num_atoms, E, e_nuc, elements, eri, coords, T, V, S);
216 end = clock();
217 serial_t = (double)(end - start) / CLOCKS_PER_SEC;
218 cout << "Serial Time: " << serial_t << endl;
219
220 input::gatherData(dataPath, num_atoms, &elements, &eri, &coords, &T, &V, &S,
221                  &e_nuc);
222 double omp_t;
223 /* int num_threads = 10; */
224 omp_set_num_threads(num_threads);
225 Eigen::setNbThreads(num_threads);
226 start = clock();
227 itime = omp_get_wtime();
228 HF(num_atoms, E, e_nuc, elements, eri, coords, T, V, S);
229 ftime = omp_get_wtime();
230 end = clock();
231 omp_t = (double)(end - start);
232 exec_time = ftime - itime;
233 double ompSpeedUp = serial_t / exec_time;
234 double eff = serial_t / (exec_time * num_threads);
235
236 cout << "Serial Time (CPU) : " << serial_t << endl;
237 cout << "OMP Time (CPU) : " << (double)(omp_t / CLOCKS_PER_SEC) << endl;
238 cout << "OMP Time (USR) : " << exec_time << endl;
239 cout << "Omp Speedup : " << ompSpeedUp << endl;
240 cout << "Parallel Efficiency: " << eff << endl;
241 }
242
243 int main(int argc, char *argv[]) {
244     printf("\nRunning: %s\n\n", argv[0]);
245     if (argc == 1) {
246         printf("You must pass a data path and number of threads like "
247              "below:\n\t./hf data/t1 4\n");
248         return 1;
249     }
250     std::string dataPath = "";
251     int numThreads = 1;
252     if (argc >= 2) {
253         dataPath = argv[1];
254         numThreads = atoi(argv[2]);
255     }
256     cout << "Data Path: " << dataPath << endl;
257     cout << "Num Threads: " << numThreads << endl;
258     cout.precision(15);

```

```

259  /* timings_parallel(dataPath, numThreads); */
260  timings(dataPath, numThreads);
261  return 0;
262 }

```

input.cpp

```

1  #include "input.hpp"
2  #include "helper.hpp"
3  #include <Eigen/Dense>
4  #include <algorithm>
5  #include <fstream>
6  #include <iostream>
7  #include <sstream>
8  #include <string>
9  #include <vector>
10
11 using namespace std;
12
13 void input::readVector(std::string fn, std::vector<std::vector<double>> **arr) {
14     std::ifstream file(fn);
15     if (!file) {
16         std::cout << "Could not open file: " << fn << std::endl;
17         return;
18     }
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::MatrixX<double>(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     std::ifstream file(fn);
87     if (!file) {
88         std::cout << "Could not open file: " << fn << std::endl;
89         return;
90     }
91     std::string line;
92     int count = 0;
93
94     while (getline(file, line)) {
95         count++;
96     }
97     file.clear();
98     file.seekg(0);
99
100    *arr = new std::vector<double>(count);
101
102    int i = 0;
103    while (getline(file, line)) {
104        std::stringstream ss(line);
105        ss >> (*arr)->at(i);
106        i++;
107    }
108    file.close();
109    return;
110 }

```



```

111
112 void input::readERI(std::string fn, std::vector<double> **arr, int n_basis) {
113     std::ifstream file(fn);
114     if (!file) {
115         std::cout << "Could not open file: " << fn << std::endl;
116         return;
117     }
118     std::string line;
119     int count = 0;
120
121     while (getline(file, line)) {
122         count++;
123     }
124     file.clear();
125     file.seekg(0);
126     /* int arrSize = */
127         /* helper::indexIJKL(n_basis - 1, n_basis - 1, n_basis - 1, n_basis - 1) + 1; */
128
129     /* cout << "arrSize: " << arrSize << endl; */
130     /* cout << "count: " << count << endl; */
131
132     *arr = new std::vector<double>(count);
133
134     int i, j, k, l;
135     double value;
136     int ijkl;
137     while (getline(file, line)) {
138         std::stringstream ss(line);
139         ss >> i >> j >> k >> l >> value;
140         ijkl = helper::indexIJKL(i, j, k, l);
141         /* std::cout << ijkl << " " << i << " " << j << " " << k << " " << l << " "
142            */
143         /* << value << std::endl; */
144         (*arr)->at(ijkl) = value;
145     }
146     file.close();
147     return;
148 }
149
150 void input::gatherData(std::string dataPath, int &num_atoms,
151     std::vector<int> **elements, std::vector<double> **eri,
152     std::vector<std::vector<double>> **coords,
153     std::vector<std::vector<double>> **T,
154     std::vector<std::vector<double>> **V,
155     std::vector<std::vector<double>> **e1,
156     std::vector<std::vector<double>> **overlap
157 ) {
158     std::string geom = dataPath + "/geom.xyz";
159     std::string eriFN = dataPath + "/eri.dat";
160     std::string TFN = dataPath + "/T.dat";
161     std::string VFN = dataPath + "/V.dat";
162     std::string e1FN = dataPath + "/e1.dat";
163     std::string overlapFN = dataPath + "/overlap.dat";
164     // Gathering Geometry
165     input::readGeometry(geom, num_atoms, elements, coords);
166     std::cout << "Number of atoms: " << num_atoms << std::endl;
167     input::printElements(*elements);
168     input::printVector(*coords);
169

```

```

170
171 // Gathering T, V, e1, overlap
172 input::readVector(TFN, T);
173 /* input::printVector(*T); */
174 input::readVector(VFN, V);
175 /* input::printVector(*V); */
176 input::readVector(e1FN, e1);
177 /* input::printVector(*e1); */
178 input::readVector(overlapFN, overlap);
179 input::readVector(eriFN, eri);
180 /* printVector(*eri); */
181 }
182 void input::gatherData(std::string dataPath, int &num_atoms,
183                       std::vector<int> **elements, std::vector<double> **eri,
184                       Eigen::MatrixXd **coords, Eigen::MatrixXd **T,
185                       Eigen::MatrixXd **V,
186                       Eigen::MatrixXd **overlap, double *enuc) {
187     std::string geom = dataPath + "/geom.xyz";
188     std::string eriFN = dataPath + "/eri.csv";
189     std::string TFN = dataPath + "/T.csv";
190     std::string VFN = dataPath + "/V.csv";
191     std::string overlapFN = dataPath + "/S.csv";
192     std::string enucFN = dataPath + "/enuc.dat";
193     // Gathering Geometry
194     input::readGeometry(geom, num_atoms, elements, coords);
195
196     // Gathering T, V, e1, overlap
197     input::readVector(TFN, T);
198     input::readVector(VFN, V);
199     input::readVector(overlapFN, overlap);
200
201     // Gathering eri
202     int n_basis = (*T)->rows();
203     cout << "n_basis: " << n_basis << endl;
204     input::readERI(eriFN, eri, n_basis);
205     input::readNumber(enucFN, *enuc);
206 }
207
208 void input::numAtoms(std::string filename, int &num_atoms) {
209     std::ifstream file(filename);
210     if (!file) {
211         std::cout << "Could not open file " << filename << std::endl;
212         return;
213     }
214     file >> num_atoms;
215 }
216
217 void input::gatherData(std::string dataPath, int &num_atoms,
218                       std::vector<int> **elements, std::vector<double> **eri,
219                       Eigen::MatrixXd **coords, Eigen::MatrixXd **T,
220                       Eigen::MatrixXd **V, Eigen::MatrixXd **e1,
221                       Eigen::MatrixXd **overlap, double *enuc) {
222     std::string geom = dataPath + "/geom.xyz";
223     std::string eriFN = dataPath + "/eri.dat";
224     std::string TFN = dataPath + "/T.dat";
225     std::string VFN = dataPath + "/V.dat";
226     std::string e1FN = dataPath + "/e1.dat";
227     std::string overlapFN = dataPath + "/overlap.dat";
228     std::string enucFN = dataPath + "/enuc.dat";

```

```

229 // Gathering Geometry
230 input::readGeometry(geom, num_atoms, elements, coords);
231
232 // Gathering T, V, e1, overlap
233 input::readVector(TFN, T);
234 input::readVector(VFN, V);
235 input::readVector(e1FN, e1);
236 input::readVector(overlapFN, overlap);
237
238 // Gathering eri
239 int n_basis = (*T)->rows();
240 cout << "n_basis: " << n_basis << endl;
241 input::readERI(eriFN, eri, n_basis);
242 input::readNumber(enucFN, *enuc);
243 }
244
245 void input::readGeometry(std::string filename, int &num_atoms,
246                         std::vector<int> **elements,
247                         Eigen::MatrixXd **coords) {
248     std::ifstream file(filename);
249     if (!file) {
250         std::cout << "Could not open file " << filename << std::endl;
251         return;
252     }
253
254     file >> num_atoms;
255     *elements = new std::vector<int>(num_atoms);
256     *coords = new Eigen::MatrixXd(num_atoms, 3);
257
258     std::string line;
259     std::getline(file, line); // read in the comment line
260
261     for (int i = 0; i < num_atoms; ++i) {
262         int el = -1;
263         double x, y, z;
264         file >> el >> x >> y >> z;
265         (*elements)->at(i) = el;
266         (**coords)(i, 0) = x;
267         (**coords)(i, 1) = y;
268         (**coords)(i, 2) = z;
269     }
270     file.close();
271     return;
272 }
273
274 void input::readGeometry(std::string filename, int &num_atoms,
275                         std::vector<int> **elements,
276                         std::vector<std::vector<double>> **coords) {
277     std::ifstream file(filename);
278     if (!file) {
279         std::cout << "Could not open file " << filename << std::endl;
280         return;
281     }
282
283     file >> num_atoms;
284     *elements = new std::vector<int>(num_atoms);
285     *coords = new std::vector<std::vector<double>>(num_atoms);
286     for (int i = 0; i < num_atoms; ++i) {
287         (*coords)->at(i) = std::vector<double>(3);

```

```

288     }
289
290     std::string line;
291     std::getline(file, line); // read in the comment line
292
293     for (int i = 0; i < num_atoms; ++i) {
294         int el = -1;
295         double x, y, z;
296         file >> el >> x >> y >> z;
297         (*elements)->at(i) = el;
298         (*coords)->at(i).at(0) = x;
299         (*coords)->at(i).at(1) = y;
300         (*coords)->at(i).at(2) = z;
301     }
302     file.close();
303     return;
304 }
305
306 void input::printVector(std::vector<std::vector<double>> *matrix) {
307     std::cout << std::endl;
308     for (int i = 0; u_int64_t(i) < matrix->size(); ++i) {
309         for (int j = 0; u_int64_t(j) < matrix->at(i).size(); ++j) {
310             std::cout.precision(12);
311             std::cout << matrix->at(i).at(j) << " ";
312         }
313         std::cout << std::endl;
314     }
315     std::cout << std::endl;
316 }
317
318 void input::printVector(std::vector<double> *matrix) {
319     std::cout << std::endl;
320     for (int i = 0; u_int64_t(i) < matrix->size(); ++i) {
321         std::cout.precision(12);
322         std::cout << matrix->at(i) << " ";
323     }
324     std::cout << std::endl;
325 }
326
327 void input::printElements(std::vector<int> *matrix) {
328     for (int i = 0; u_int64_t(i) < matrix->size(); ++i) {
329         std::cout << matrix->at(i) << " ";
330         std::cout << std::endl;
331     }
332 }
333
334 void input::readNumber(std::string filename, double &number) {
335     std::ifstream file(filename);
336     if (!file) {
337         std::cout << "Could not open file " << filename << std::endl;
338         return;
339     }
340     file >> number;
341     file.close();
342 }

```

helper.cpp

```

1  #include "helper.hpp"
2  #include "stdio.h"
3  #include <Eigen/Dense>
4  #include <iostream>
5  #include <vector>
6
7  using namespace Eigen;
8  using namespace std;
9
10 void helper::orthoS(Eigen::MatrixXd *S, Eigen::MatrixXd *S12) {
11     // Diagonalize S
12     Eigen::SelfAdjointEigenSolver<Eigen::MatrixXd> es(*S);
13     /* #pragma omp parallel public(es, LAMBDA) */
14     Eigen::MatrixXd LAMBDA = es.eigenvalues().asDiagonal();
15     Eigen::MatrixXd U = es.eigenvectors();
16     // Invert D
17
18     // TODO: Parallelize
19 #pragma omp parallel for
20     for (int i = 0; i < LAMBDA.rows(); i++) {
21         LAMBDA(i, i) = 1 / sqrt(LAMBDA(i, i));
22     }
23     // Calculate X
24     *S12 = U * LAMBDA * U.transpose();
25 }
26
27 void helper::initialFockMatrix(Eigen::MatrixXd *X, Eigen::MatrixXd *H,
28                               Eigen::MatrixXd *F) {
29     // Calculate F
30     *F = (*X).transpose() * *H * (*X);
31 }
32
33 void helper::getC_0_prime(Eigen::MatrixXd *F, Eigen::MatrixXd *C) {
34     // Diagonalize F
35     SelfAdjointEigenSolver<MatrixXd> eigensolver(*F);
36     if (eigensolver.info() != Success)
37         abort(); // check for errors
38     *C = eigensolver.eigenvectors();
39 }
40
41 void helper::computeEnergy(Eigen::MatrixXd *D, Eigen::MatrixXd *H,
42                            Eigen::MatrixXd *F, double *E) {
43     // Calculate E
44     *E = 0;
45     *E = (*D).cwiseProduct((*H) + (*F)).sum();
46 }
47
48 void helper::getNumberOfElectrons(int num_atoms, std::vector<int> *elements,
49                                   int *num_electrons) {
50     // Calculate number of electrons
51     *num_electrons = 0;
52     for (int i = 0; i < num_atoms; i++) {
53         *num_electrons += elements->at(i);
54     }
55 }
56
57 int helper::indexIJKL(int i, int j, int k, int l) {
58     if (j > i) {
59         std::swap(i, j);

```

```

60     }
61     if (l > k) {
62         std::swap(k, l);
63     }
64     int ij = i * (i + 1) / 2 + j;
65     int kl = k * (k + 1) / 2 + l;
66     if (ij < kl) {
67         std::swap(ij, kl);
68     }
69     int ijk1 = ij * (ij + 1) / 2 + kl;
70     return ijk1;
71 }
72
73 void helper::updateDensityMatrix(Eigen::MatrixXd *C, Eigen::MatrixXd *D,
74                                 int num_electrons) {
75     // Calculate D
76     // TODO: Parallelize
77     #pragma omp parallel for
78     for (int i = 0; i < C->rows(); i++) {
79         for (int j = 0; j < C->rows(); j++) {
80             (*D)(i, j) = 0;
81             for (int k = 0; k < num_electrons / 2; k++) {
82                 (*D)(i, j) += (*C)(i, k) * (*C)(j, k);
83             }
84         }
85     }
86 }
87
88 void helper::updateFockMatrix(Eigen::MatrixXd *H, Eigen::MatrixXd *D,
89                               Eigen::MatrixXd *F, std::vector<double> *eri) {
90     // Update Fock Matrix
91     *F = *H;
92     #pragma omp parallel for
93     for (int mu = 0; mu < H->rows(); mu++) {
94         for (int nu = 0; nu < H->cols(); nu++) {
95             for (int rho = 0; rho < H->rows(); rho++) {
96                 for (int sig = 0; sig < H->cols(); sig++) {
97                     (*F)(mu, nu) += (*D)(rho, sig) *
98                                     (2 * eri->at(helper::indexIJKL(mu, nu, rho, sig)) -
99                                     eri->at(helper::indexIJKL(mu, rho, nu, sig)));
100                 }
101             }
102         }
103     }
104 }
105
106 void helper::SCF(std::vector<double> *eri, Eigen::MatrixXd *S_12,
107                 Eigen::MatrixXd *H, Eigen::MatrixXd *F, Eigen::MatrixXd *C,
108                 Eigen::MatrixXd *D, Eigen::MatrixXd *C_0_prime,
109                 int num_electrons, double *E, double e_nuc, double t1,
110                 double t2) {
111     // Calculate SCF
112
113     bool converged = false;
114     double E2 = 0;
115     int iter = 0, max_iter = 100;
116     while (!converged) {
117         // Update Fock Matrix
118         helper::updateFockMatrix(H, D, F, eri);

```

```

119  /* cout << endl <<"F Matrix: " << endl << endl <<*F << endl; */
120  /* cout << endl <<"E: " << endl << endl <<*E << endl; */
121  helper::computeEnergy(D, H, F, E);
122  *F = (*S_12).transpose() * *F * (*S_12);
123
124  helper::getC_0_prime(F, C_0_prime);
125  /* cout << endl <<"C_0_prime Matrix: " << endl <<endl << *C_0_prime << endl;
126  */
127
128  *C = (*S_12) * (*C_0_prime);
129  /* cout << endl <<"C Matrix: " << endl <<endl << *C << endl; */
130  helper::updateDensityMatrix(C, D, num_electrons);
131  /* cout << endl <<"D Matrix: " << endl << endl <<*D << endl; */
132
133  cout << "iter: " << iter << " Energy: " << *E << " Delta E: " << (*E - E2)
134  << endl;
135  if (abs(*E - E2) < t1) {
136      converged = true;
137  } else if (iter > max_iter) {
138      cout << "Max iterations reached" << endl;
139      converged = true;
140  } else {
141      E2 = *E;
142      iter++;
143  }
144  }
145  }

```

main.py

```

1  import psi4
2  import numpy as np
3  import pandas as pd
4  from qm_tools_aw import tools
5  import qcelestial as qcel
6
7
8  def psi4_compute(mol, outdata="t2"):
9      with open(f"{outdata}/geom.xyz", 'w') as f:
10          n = mol.count('\n')
11          f.write(f"{n}\n\n")
12          f.write(mol)
13
14      mol = psi4.geometry(mol)
15      psi4.set_memory('4 GB')
16      psi4.set_num_threads(10)
17      psi4.core.set_output_file(f'{outdata}/output.dat', False)
18      psi4.set_options({"basis": "aug-cc-pvdz"})
19      wfn = psi4.core.Wavefunction.build(mol,
20                                         psi4.core.get_global_option("basis"))
21      mints = psi4.core.MintsHelper(wfn.basisset())
22      S = np.asarray(mints.ao_overlap())
23      np.savetxt(f"{outdata}/S.csv", S, delimiter=" ")
24      T = np.asarray(mints.ao_potential())
25      np.savetxt(f"{outdata}/T.csv", T, delimiter=" ")
26      V = np.asarray(mints.ao_kinetic())
27      np.savetxt(f"{outdata}/V.csv", V, delimiter=" ")
28

```

```

29 I = np.asarray(mints.ao_eri())
30 nbf = len(I)
31 print(f"{nbf = }")
32
33 with open(f"{outdata}/eri.csv", 'w') as f:
34     for i in range(nbf):
35         for j in range(i + 1):
36             for k in range(i + 1):
37                 for l in range(k + 1):
38                     line = f"{i} {j} {k} {l} {I[i,j,k,l]}\n"
39                     f.write(line)
40
41 e = psi4.energy("HF/aug-cc-pvdz")
42 print(f"{e =}")
43 return
44
45 def find_geoms(size=10) -> None:
46     """
47     find_geoms
48     """
49     mol = psi4.geometry("""
50 0 1
51 8    0.000000000000    0.000000000000    -0.071151380605
52 1    0.000000000000    0.757939245855    0.564612021746
53 1    0.000000000000    -0.757939245855    0.564612021746
54 """)
55 df = pd.read_pickle("schr.pkl")
56 print(df)
57 for n, r in df.iterrows():
58     if len(r['monAs']) == size:
59         mmA = r['Geometry'][r['monAs'], :]
60         # tools.print_cartesians_pos_carts(mmA[:,0], mmA[:,1:])
61         return tools.print_cartesians_pos_carts(mmA[:,0], mmA[:,1:])
62
63 def benzene():
64     return """
65 6    1.5000000000    -1.8000000000    -1.3915000000
66 6    2.7050743494    -1.8000000000    -0.6957500000
67 6    2.7050743494    -1.8000000000    0.6957500000
68 6    1.5000000000    -1.8000000000    1.3915000000
69 6    0.2949256506    -1.8000000000    0.6957500000
70 6    0.2949256506    -1.8000000000    -0.6957500000
71 1    1.5000000000    -1.8000000000    -2.4715000000
72 1    3.6403817855    -1.8000000000    -1.2357500000
73 1    3.6403817855    -1.8000000000    1.2357500000
74 1    1.5000000000    -1.8000000000    2.4715000000
75 1    -0.6403817855    -1.8000000000    1.2357500000
76 1    -0.6403817855    -1.8000000000    -1.2357500000
77 """
78
79 def main():
80     d = find_geoms(6) # Ethene
81     # return
82     # d = benzene()
83     psi4.compute(d, outdata="t3")
84     return
85
86
87 if __name__ == "__main__":

```


Output

The initial output is from using the “Coding Strategy #1” water geometry with the STO-3G basis set using the HF_og function. Additionally, I performed timing tests for speedup and efficiency for both Water HF/STO-3G and Ethene HF/aug-cc-pVDZ as can be seen below.

HF_og()

```

1  -- The CXX compiler identification is GNU 12.2.0
2  -- Checking whether CXX compiler has -isysroot
3  -- Checking whether CXX compiler has -isysroot - yes
4  -- Checking whether CXX compiler supports OSX deployment target flag
5  -- Checking whether CXX compiler supports OSX deployment target flag - yes
6  -- Detecting CXX compiler ABI info
7  -- Detecting CXX compiler ABI info - done
8  -- Check for working CXX compiler: /usr/local/bin/g++-12 - skipped
9  -- Detecting CXX compile features
10 -- Detecting CXX compile features - done
11 -- Found OpenMP_CXX: -fopenmp (found version "4.5")
12 -- Found OpenMP: TRUE (found version "4.5")
13 -- Looking for sgemm_
14 -- Looking for sgemm_ - not found
15 -- Performing Test CMAKE_HAVE_LIBC_PTHREAD
16 -- Performing Test CMAKE_HAVE_LIBC_PTHREAD - Success
17 -- Found Threads: TRUE
18 -- Looking for dgemm_
19 -- Looking for dgemm_ - found
20 -- Found BLAS: /Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/
    ↳ Developer/SDKs/MacOSX13.3.sdk/System/Library/Frameworks/Accelerate.framework
21 -- Looking for cheev_
22 -- Looking for cheev_ - found
23 -- Found LAPACK: /Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/
    ↳ Developer/SDKs/MacOSX13.3.sdk/System/Library/Frameworks/Accelerate.framework;-lm
    ↳ ;-ldl
24 -- Found MPI_CXX: /Users/austinwallace/miniconda3/envs/qcn/lib/libmpicxx.dylib (found
    ↳ version "3.1")
25 -- Found MPI: TRUE (found version "3.1")
26 -- Found OpenMP_CXX: -fopenmp (found version "4.5")
27 -- Configuring done
28 -- Generating done
29 -- Build files have been written to: /Users/austinwallace/gits/HF/cpp/build
30 [ 25%] Building CXX object src/CMakeFiles/hf.dir/hf.cpp.o
31 [ 50%] Building CXX object src/CMakeFiles/hf.dir/input.cpp.o
32 [ 75%] Building CXX object src/CMakeFiles/hf.dir/helper.cpp.o
33 [100%] Linking CXX executable hf
34 ld: warning: directory not found for option '-L/usr/include/eigen3'
35 [100%] Built target hf
36 n_basis: 7
37 arrSize: 406
38 count: 406
39 e_nuc: 8.90771
40
41 H Matrix:
42
43    -32.6851    -7.60432         0         0 -0.0186797    -1.6196    -1.6196

```

```

44  -7.60432  -9.30206  0  0 -0.22216  -3.54321  -3.54321
45  0  0 -7.43084  0  0  0  0
46  0  0  0 -7.56702  0 -1.89086  1.89086
47 -0.0186797 -0.22216  0  0 -7.52666 -1.65879 -1.65879
48 -1.6196 -3.54321  0 -1.89086 -1.65879 -4.95649 -1.56026
49 -1.6196 -3.54321  0  1.89086 -1.65879 -1.56026 -4.95649
50
51  S Matrix:
52
53  1  0.236704  0  0 -0 0.0500137 0.0500137
54  0.236704  1  0  0 -0 0.453995 0.453995
55  0  0  1  0  0  0  0
56  0  0  0  1  0 0.292739 -0.292739
57  0  -0  0  0  1 0.245551 0.245551
58  0.0500137 0.453995  0 0.292739 0.245551  1 0.251002
59  0.0500137 0.453995  0 -0.292739 0.245551 0.251002  1
60
61  S_12 Matrix:
62
63  1.02406  -0.141659 -3.22048e-18  0 -0.0100026  0.0212116
64  ↪ 0.0212116
65  -0.141659  1.22192 -2.96899e-17  2.47692e-18  0.105912  -0.275658
66  ↪ -0.275658
67  -3.22048e-18 -2.96899e-17  1  4.23252e-17  1.97909e-17  1.30612e-16  8.2861
68  ↪ e-17
69  0  2.47692e-18  4.23252e-17  1.09816 -1.62843e-16  -0.213038
70  ↪ 0.213038
71  -0.0100026  0.105912  1.97909e-17 -1.62843e-16  1.05609  -0.146486
72  ↪ -0.146486
73  0.0212116  -0.275658  1.30612e-16  -0.213038  -0.146486  1.19048
74  ↪ -0.0903463
75  0.0212116  -0.275658  8.2861e-17  0.213038  -0.146486  -0.0903463
76  ↪ 1.19048
77
78  F Matrix:
79
80  -32.3609  -2.78101  5.24172e-17  1.11022e-16  0.0165515  -0.273188
81  ↪ -0.273188
82  -2.78101  -8.32926  -5.1327e-17  4.44089e-16  -0.281188  -0.481149
83  ↪ -0.481149
84  5.24172e-17  -5.1327e-17  -7.43084 -6.96731e-16 -5.38842e-16 -1.57009e-15 -9.77471
85  ↪ e-16
86  1.8735e-16  1.66533e-16 -6.96731e-16  -7.66432  2.35922e-15  -0.134212
87  ↪ 0.134212
88  0.0165515  -0.281188 -5.38842e-16  2.44249e-15  -7.57829  -0.146808
89  ↪ -0.146808
90  -0.273188  -0.481149 -1.57009e-15  -0.134212  -0.146808  -4.24477
91  ↪ -0.0501421
92  -0.273188  -0.481149 -9.77471e-16  0.134212  -0.146808  -0.0501421
93  ↪ -4.24477
94
95  C_0_prime Matrix:
96
97  -0.993361  -0.104697 -6.45607e-16  0.0476199 -7.98146e-15  4.0881e-16
98  ↪ 0.00222819
99  -0.113883  0.887058  5.797e-15  -0.417863  6.99798e-14 -2.99761e-14
100 ↪ -0.159843
101  3.15273e-19  1.64226e-15  4.80473e-15  1.71258e-13  1 -1.14056e-16 -7.14186
102 ↪ e-16

```

```

86   9.7715e-18 -2.46936e-15      0.998516  8.28671e-15 -4.55392e-15      0.0544598 -9.44874
      ↪ e-15
87 -0.000754957      0.418666 -6.32026e-15      0.906926 -1.55863e-13 -8.64117e-15
      ↪ -0.0469432
88 -0.0114923      0.115943      0.0385089 -0.0174439  3.15953e-15      -0.706057
      ↪ 0.697224
89 -0.0114923      0.115943      -0.0385089 -0.0174439  3.58246e-15      0.706057
      ↪ 0.697224
90
91 C Matrix:
92
93   -1.00161      -0.232145 -1.42291e-15      0.0981483 -1.6388e-14  1.02331e-14
      ↪ 0.054973
94   0.00781923      1.07916  6.54511e-15      -0.411669  6.82443e-14 -1.08663e-13
      ↪ -0.584993
95   4.4273e-18      1.6493e-15  4.84883e-15  1.71285e-13      1 -1.45466e-16 -5.61539
      ↪ e-16
96 -4.33681e-18 -2.62637e-15      1.08012  8.75992e-15 -4.86851e-15      0.360639 -6.61415
      ↪ e-14
97   0.000444284      0.503178 -6.19296e-15      0.918173 -1.58082e-13 -5.01404e-14
      ↪ -0.270795
98  -0.00221056      -0.180521      -0.163398 -0.0358456  7.9105e-15      -0.915938
      ↪ 0.818024
99  -0.00221056      -0.180521      0.163398 -0.0358456  6.46414e-15      0.915938
      ↪ 0.818024
100
101 D Matrix:
102
103   1.06675      -0.298759  3.60303e-17 -6.31019e-17      -0.0271382  0.0406029
      ↪ 0.0406029
104  -0.298759      1.33412 -4.88497e-16  6.29026e-16      0.165031 -0.180072
      ↪ -0.180072
105  3.60303e-17 -4.88497e-16      1  3.68824e-16  1.73216e-17  6.80666e-16  8.18881
      ↪ e-16
106 -6.31019e-17  6.29026e-16  3.68824e-16      1.16667  3.24209e-17      -0.17649
      ↪ 0.17649
107  -0.0271382      0.165031  1.73216e-17  3.24209e-17      1.09623 -0.123747
      ↪ -0.123747
108   0.0406029      -0.180072  6.80666e-16      -0.17649 -0.123747  0.0605765
      ↪ 0.00717853
109   0.0406029      -0.180072  8.18881e-16      0.17649 -0.123747  0.00717853
      ↪ 0.0605765
110 iter: 0 Energy: -82.1486 Delta E: -82.1486
111 iter: 1 Energy: -83.8388 Delta E: -1.69013
112 iter: 2 Energy: -83.8722 Delta E: -0.0333948
113 iter: 3 Energy: -83.8734 Delta E: -0.00128959
114 iter: 4 Energy: -83.8736 Delta E: -0.000137273
115 iter: 5 Energy: -83.8736 Delta E: -2.36545e-05
116 iter: 6 Energy: -83.8736 Delta E: -4.48631e-06
117 iter: 7 Energy: -83.8736 Delta E: -8.72434e-07
118 iter: 8 Energy: -83.8736 Delta E: -1.70848e-07
119 iter: 9 Energy: -83.8736 Delta E: -3.35253e-08
120 iter: 10 Energy: -83.8736 Delta E: -6.58248e-09
121
122 Final HF Energy: -74.9659010585405
123
124 Freed Memory
125 Serial Time: 0.011638

```

Water HF/STO-3G

```
1 \nTiming for data/t1\n
2 ./hf data/t1 1
3
4 Running: ./hf
5
6 Data Path: data/t1
7 Num Threads: 1
8 n_basis: 7
9 iter: 0 Energy: -82.1486 Delta E: -82.1486
10 iter: 1 Energy: -83.8388 Delta E: -1.69013
11 iter: 2 Energy: -83.8722 Delta E: -0.0333948
12 iter: 3 Energy: -83.8734 Delta E: -0.00128959
13 iter: 4 Energy: -83.8736 Delta E: -0.000137273
14 iter: 5 Energy: -83.8736 Delta E: -2.36545e-05
15 iter: 6 Energy: -83.8736 Delta E: -4.48631e-06
16 iter: 7 Energy: -83.8736 Delta E: -8.72434e-07
17 iter: 8 Energy: -83.8736 Delta E: -1.70848e-07
18 iter: 9 Energy: -83.8736 Delta E: -3.35253e-08
19 iter: 10 Energy: -83.8736 Delta E: -6.58248e-09
20
21 Final HF Energy: -74.9659010585405
22
23 Freed Memory
24 Time (CPU) : 0.008954
25 Time (USR) : 0.00894462922587991
26
27 ./hf data/t1 2
28
29 Running: ./hf
30
31 Data Path: data/t1
32 Num Threads: 2
33 n_basis: 7
34 iter: 0 Energy: -82.1486 Delta E: -82.1486
35 iter: 1 Energy: -83.8388 Delta E: -1.69013
36 iter: 2 Energy: -83.8722 Delta E: -0.0333948
37 iter: 3 Energy: -83.8734 Delta E: -0.00128959
38 iter: 4 Energy: -83.8736 Delta E: -0.000137273
39 iter: 5 Energy: -83.8736 Delta E: -2.36545e-05
40 iter: 6 Energy: -83.8736 Delta E: -4.48631e-06
41 iter: 7 Energy: -83.8736 Delta E: -8.72434e-07
42 iter: 8 Energy: -83.8736 Delta E: -1.70848e-07
43 iter: 9 Energy: -83.8736 Delta E: -3.35253e-08
44 iter: 10 Energy: -83.8736 Delta E: -6.58248e-09
45
46 Final HF Energy: -74.9659010585405
47
48 Freed Memory
49 Time (CPU) : 0.012446
50 Time (USR) : 0.00687378598377109
51
52 ./hf data/t1 4
53
54 Running: ./hf
55
56 Data Path: data/t1
57 Num Threads: 4
```

```

58 n_basis: 7
59 iter: 0 Energy: -82.1486 Delta E: -82.1486
60 iter: 1 Energy: -83.8388 Delta E: -1.69013
61 iter: 2 Energy: -83.8722 Delta E: -0.0333948
62 iter: 3 Energy: -83.8734 Delta E: -0.00128959
63 iter: 4 Energy: -83.8736 Delta E: -0.000137273
64 iter: 5 Energy: -83.8736 Delta E: -2.36545e-05
65 iter: 6 Energy: -83.8736 Delta E: -4.48631e-06
66 iter: 7 Energy: -83.8736 Delta E: -8.72434e-07
67 iter: 8 Energy: -83.8736 Delta E: -1.70848e-07
68 iter: 9 Energy: -83.8736 Delta E: -3.35253e-08
69 iter: 10 Energy: -83.8736 Delta E: -6.58248e-09
70
71 Final HF Energy: -74.9659010585405
72
73 Freed Memory
74 Time (CPU) : 0.022395
75 Time (USR) : 0.00679890578612685
76
77 ./hf data/t1 6
78
79 Running: ./hf
80
81 Data Path: data/t1
82 Num Threads: 6
83 n_basis: 7
84 iter: 0 Energy: -82.1486 Delta E: -82.1486
85 iter: 1 Energy: -83.8388 Delta E: -1.69013
86 iter: 2 Energy: -83.8722 Delta E: -0.0333948
87 iter: 3 Energy: -83.8734 Delta E: -0.00128959
88 iter: 4 Energy: -83.8736 Delta E: -0.000137273
89 iter: 5 Energy: -83.8736 Delta E: -2.36545e-05
90 iter: 6 Energy: -83.8736 Delta E: -4.48631e-06
91 iter: 7 Energy: -83.8736 Delta E: -8.72434e-07
92 iter: 8 Energy: -83.8736 Delta E: -1.70848e-07
93 iter: 9 Energy: -83.8736 Delta E: -3.35253e-08
94 iter: 10 Energy: -83.8736 Delta E: -6.58248e-09
95
96 Final HF Energy: -74.9659010585405
97
98 Freed Memory
99 Time (CPU) : 0.020431
100 Time (USR) : 0.00625808583572507
101
102 ./hf data/t1 8
103
104 Running: ./hf
105
106 Data Path: data/t1
107 Num Threads: 8
108 n_basis: 7
109 iter: 0 Energy: -82.1486 Delta E: -82.1486
110 iter: 1 Energy: -83.8388 Delta E: -1.69013
111 iter: 2 Energy: -83.8722 Delta E: -0.0333948
112 iter: 3 Energy: -83.8734 Delta E: -0.00128959
113 iter: 4 Energy: -83.8736 Delta E: -0.000137273
114 iter: 5 Energy: -83.8736 Delta E: -2.36545e-05
115 iter: 6 Energy: -83.8736 Delta E: -4.48631e-06
116 iter: 7 Energy: -83.8736 Delta E: -8.72434e-07

```

```

117 iter: 8 Energy: -83.8736 Delta E: -1.70848e-07
118 iter: 9 Energy: -83.8736 Delta E: -3.35253e-08
119 iter: 10 Energy: -83.8736 Delta E: -6.58248e-09
120
121 Final HF Energy: -74.9659010585405
122
123 Freed Memory
124 Time (CPU) : 0.030145
125 Time (USR) : 0.00609015300869942
126
127 ./hf data/t1 10
128
129 Running: ./hf
130
131 Data Path: data/t1
132 Num Threads: 10
133 n_basis: 7
134 iter: 0 Energy: -82.1486 Delta E: -82.1486
135 iter: 1 Energy: -83.8388 Delta E: -1.69013
136 iter: 2 Energy: -83.8722 Delta E: -0.0333948
137 iter: 3 Energy: -83.8734 Delta E: -0.00128959
138 iter: 4 Energy: -83.8736 Delta E: -0.000137273
139 iter: 5 Energy: -83.8736 Delta E: -2.36545e-05
140 iter: 6 Energy: -83.8736 Delta E: -4.48631e-06
141 iter: 7 Energy: -83.8736 Delta E: -8.72434e-07
142 iter: 8 Energy: -83.8736 Delta E: -1.70848e-07
143 iter: 9 Energy: -83.8736 Delta E: -3.35253e-08
144 iter: 10 Energy: -83.8736 Delta E: -6.58248e-09
145
146 Final HF Energy: -74.9659010585405
147
148 Freed Memory
149 Time (CPU) : 0.045239
150 Time (USR) : 0.0056780893355608

```

Ethene HF/aug-cc-pVDZ

```

1 Timing for data/t3
2 ./hf data/t3 1
3
4 Running: ./hf
5
6 Data Path: data/t3
7 Num Threads: 1
8 n_basis: 82
9 iter: 0 Energy: -97.0923 Delta E: -97.0923
10 iter: 1 Energy: -98.28 Delta E: -1.18774
11 iter: 2 Energy: -107.398 Delta E: -9.11804
12 iter: 3 Energy: -105.85 Delta E: 1.5485
13 iter: 4 Energy: -109.396 Delta E: -3.54656
14 iter: 5 Energy: -107.579 Delta E: 1.81723
15 iter: 6 Energy: -110.004 Delta E: -2.42545
16 iter: 7 Energy: -110.756 Delta E: -0.75169
17 iter: 8 Energy: -111.276 Delta E: -0.52038
18 iter: 9 Energy: -111.42 Delta E: -0.143198
19 iter: 10 Energy: -111.448 Delta E: -0.0283181
20 iter: 11 Energy: -111.453 Delta E: -0.00491031
21 iter: 12 Energy: -111.454 Delta E: -0.000809715

```

```

22 iter: 13 Energy: -111.454 Delta E: -0.000131563
23 iter: 14 Energy: -111.454 Delta E: -2.12882e-05
24 iter: 15 Energy: -111.454 Delta E: -3.44269e-06
25 iter: 16 Energy: -111.454 Delta E: -5.57065e-07
26 iter: 17 Energy: -111.454 Delta E: -9.02316e-08
27 iter: 18 Energy: -111.454 Delta E: -1.46336e-08
28 iter: 19 Energy: -111.454 Delta E: -2.37615e-09
29
30 Final HF Energy: -78.0435868316762
31
32 Freed Memory
33 Time (CPU) : 196.674979
34 Time (USR) : 196.682998436969
35
36 ./hf data/t3 2
37
38 Running: ./hf
39
40 Data Path: data/t3
41 Num Threads: 2
42 n_basis: 82
43 iter: 0 Energy: -97.0923 Delta E: -97.0923
44 iter: 1 Energy: -98.28 Delta E: -1.18774
45 iter: 2 Energy: -107.398 Delta E: -9.11804
46 iter: 3 Energy: -105.85 Delta E: 1.5485
47 iter: 4 Energy: -109.396 Delta E: -3.54656
48 iter: 5 Energy: -107.579 Delta E: 1.81723
49 iter: 6 Energy: -110.004 Delta E: -2.42545
50 iter: 7 Energy: -110.756 Delta E: -0.75169
51 iter: 8 Energy: -111.276 Delta E: -0.52038
52 iter: 9 Energy: -111.42 Delta E: -0.143198
53 iter: 10 Energy: -111.448 Delta E: -0.0283181
54 iter: 11 Energy: -111.453 Delta E: -0.00491031
55 iter: 12 Energy: -111.454 Delta E: -0.000809715
56 iter: 13 Energy: -111.454 Delta E: -0.000131563
57 iter: 14 Energy: -111.454 Delta E: -2.12882e-05
58 iter: 15 Energy: -111.454 Delta E: -3.44269e-06
59 iter: 16 Energy: -111.454 Delta E: -5.57065e-07
60 iter: 17 Energy: -111.454 Delta E: -9.02316e-08
61 iter: 18 Energy: -111.454 Delta E: -1.46336e-08
62 iter: 19 Energy: -111.454 Delta E: -2.37615e-09
63
64 Final HF Energy: -78.0435868316762
65
66 Freed Memory
67 Time (CPU) : 200.002911
68 Time (USR) : 106.574036244769
69
70 ./hf data/t3 4
71
72 Running: ./hf
73
74 Data Path: data/t3
75 Num Threads: 4
76 n_basis: 82
77 iter: 0 Energy: -97.0923 Delta E: -97.0923
78 iter: 1 Energy: -98.28 Delta E: -1.18774
79 iter: 2 Energy: -107.398 Delta E: -9.11804
80 iter: 3 Energy: -105.85 Delta E: 1.5485

```

```

81 iter: 4 Energy: -109.396 Delta E: -3.54656
82 iter: 5 Energy: -107.579 Delta E: 1.81723
83 iter: 6 Energy: -110.004 Delta E: -2.42545
84 iter: 7 Energy: -110.756 Delta E: -0.75169
85 iter: 8 Energy: -111.276 Delta E: -0.52038
86 iter: 9 Energy: -111.42 Delta E: -0.143198
87 iter: 10 Energy: -111.448 Delta E: -0.0283181
88 iter: 11 Energy: -111.453 Delta E: -0.00491031
89 iter: 12 Energy: -111.454 Delta E: -0.000809715
90 iter: 13 Energy: -111.454 Delta E: -0.000131563
91 iter: 14 Energy: -111.454 Delta E: -2.12882e-05
92 iter: 15 Energy: -111.454 Delta E: -3.44269e-06
93 iter: 16 Energy: -111.454 Delta E: -5.57065e-07
94 iter: 17 Energy: -111.454 Delta E: -9.02316e-08
95 iter: 18 Energy: -111.454 Delta E: -1.46336e-08
96 iter: 19 Energy: -111.454 Delta E: -2.37615e-09
97
98 Final HF Energy: -78.0435868316762
99
100 Freed Memory
101 Time (CPU) : 207.655153
102 Time (USR) : 60.0388427260332
103
104 ./hf data/t3 6
105
106 Running: ./hf
107
108 Data Path: data/t3
109 Num Threads: 6
110 n_basis: 82
111 iter: 0 Energy: -97.0923 Delta E: -97.0923
112 iter: 1 Energy: -98.28 Delta E: -1.18774
113 iter: 2 Energy: -107.398 Delta E: -9.11804
114 iter: 3 Energy: -105.85 Delta E: 1.5485
115 iter: 4 Energy: -109.396 Delta E: -3.54656
116 iter: 5 Energy: -107.579 Delta E: 1.81723
117 iter: 6 Energy: -110.004 Delta E: -2.42545
118 iter: 7 Energy: -110.756 Delta E: -0.75169
119 iter: 8 Energy: -111.276 Delta E: -0.52038
120 iter: 9 Energy: -111.42 Delta E: -0.143198
121 iter: 10 Energy: -111.448 Delta E: -0.0283181
122 iter: 11 Energy: -111.453 Delta E: -0.00491031
123 iter: 12 Energy: -111.454 Delta E: -0.000809715
124 iter: 13 Energy: -111.454 Delta E: -0.000131563
125 iter: 14 Energy: -111.454 Delta E: -2.12882e-05
126 iter: 15 Energy: -111.454 Delta E: -3.44269e-06
127 iter: 16 Energy: -111.454 Delta E: -5.57065e-07
128 iter: 17 Energy: -111.454 Delta E: -9.02316e-08
129 iter: 18 Energy: -111.454 Delta E: -1.46336e-08
130 iter: 19 Energy: -111.454 Delta E: -2.37615e-09
131
132 Final HF Energy: -78.0435868316762
133
134 Freed Memory
135 Time (CPU) : 213.795112
136 Time (USR) : 42.3328502173536
137
138 ./hf data/t3 8
139

```



```

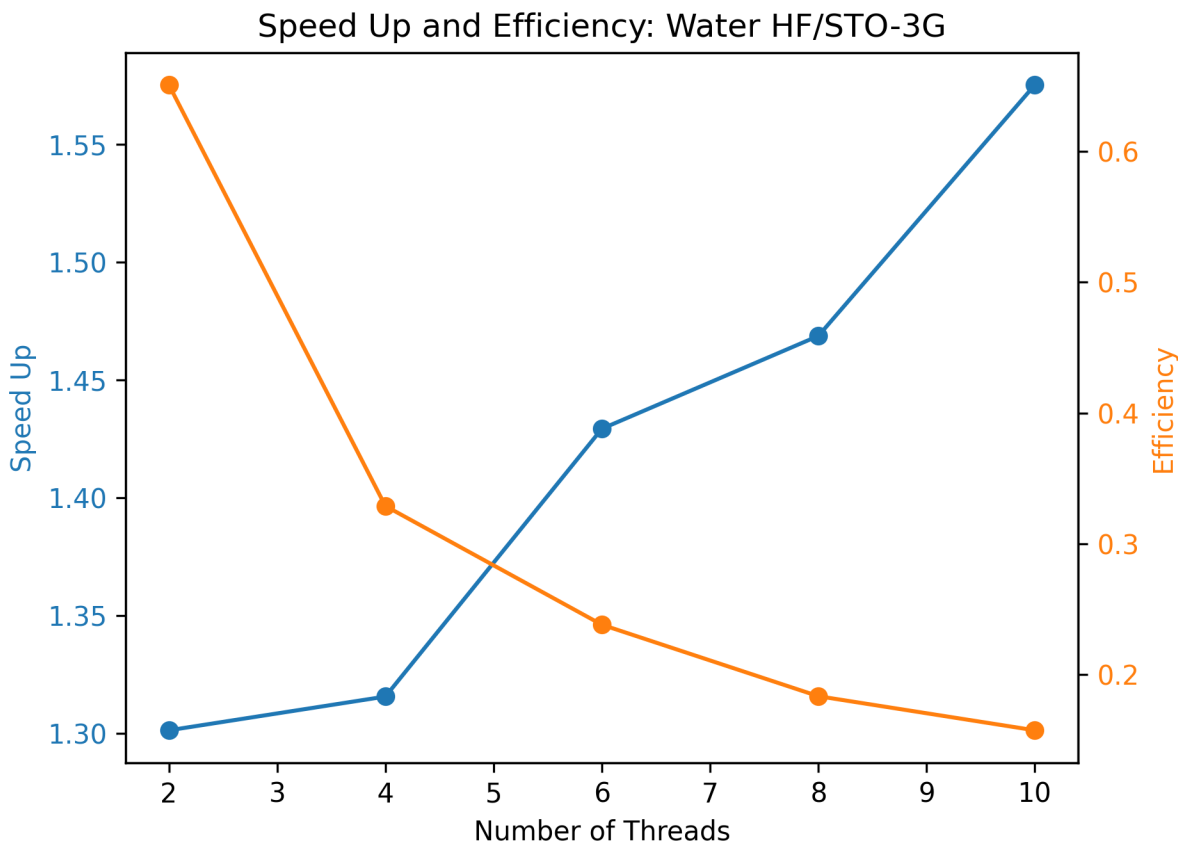
140 Running: ./hf
141
142 Data Path: data/t3
143 Num Threads: 8
144 n_basis: 82
145 iter: 0 Energy: -97.0923 Delta E: -97.0923
146 iter: 1 Energy: -98.28 Delta E: -1.18774
147 iter: 2 Energy: -107.398 Delta E: -9.11804
148 iter: 3 Energy: -105.85 Delta E: 1.5485
149 iter: 4 Energy: -109.396 Delta E: -3.54656
150 iter: 5 Energy: -107.579 Delta E: 1.81723
151 iter: 6 Energy: -110.004 Delta E: -2.42545
152 iter: 7 Energy: -110.756 Delta E: -0.75169
153 iter: 8 Energy: -111.276 Delta E: -0.52038
154 iter: 9 Energy: -111.42 Delta E: -0.143198
155 iter: 10 Energy: -111.448 Delta E: -0.0283181
156 iter: 11 Energy: -111.453 Delta E: -0.00491031
157 iter: 12 Energy: -111.454 Delta E: -0.000809715
158 iter: 13 Energy: -111.454 Delta E: -0.000131563
159 iter: 14 Energy: -111.454 Delta E: -2.12882e-05
160 iter: 15 Energy: -111.454 Delta E: -3.44269e-06
161 iter: 16 Energy: -111.454 Delta E: -5.57065e-07
162 iter: 17 Energy: -111.454 Delta E: -9.02316e-08
163 iter: 18 Energy: -111.454 Delta E: -1.46336e-08
164 iter: 19 Energy: -111.454 Delta E: -2.37615e-09
165
166 Final HF Energy: -78.0435868316762
167
168 Freed Memory
169 Time (CPU) : 218.198771
170 Time (USR) : 34.0461984351277
171
172 ./hf data/t3 10
173
174 Running: ./hf
175
176 Data Path: data/t3
177 Num Threads: 10
178 n_basis: 82
179 iter: 0 Energy: -97.0923 Delta E: -97.0923
180 iter: 1 Energy: -98.28 Delta E: -1.18774
181 iter: 2 Energy: -107.398 Delta E: -9.11804
182 iter: 3 Energy: -105.85 Delta E: 1.5485
183 iter: 4 Energy: -109.396 Delta E: -3.54656
184 iter: 5 Energy: -107.579 Delta E: 1.81723
185 iter: 6 Energy: -110.004 Delta E: -2.42545
186 iter: 7 Energy: -110.756 Delta E: -0.75169
187 iter: 8 Energy: -111.276 Delta E: -0.52038
188 iter: 9 Energy: -111.42 Delta E: -0.143198
189 iter: 10 Energy: -111.448 Delta E: -0.0283181
190 iter: 11 Energy: -111.453 Delta E: -0.00491031
191 iter: 12 Energy: -111.454 Delta E: -0.000809715
192 iter: 13 Energy: -111.454 Delta E: -0.000131563
193 iter: 14 Energy: -111.454 Delta E: -2.12882e-05
194 iter: 15 Energy: -111.454 Delta E: -3.44269e-06
195 iter: 16 Energy: -111.454 Delta E: -5.57065e-07
196 iter: 17 Energy: -111.454 Delta E: -9.02316e-08
197 iter: 18 Energy: -111.454 Delta E: -1.46336e-08
198 iter: 19 Energy: -111.454 Delta E: -2.37615e-09

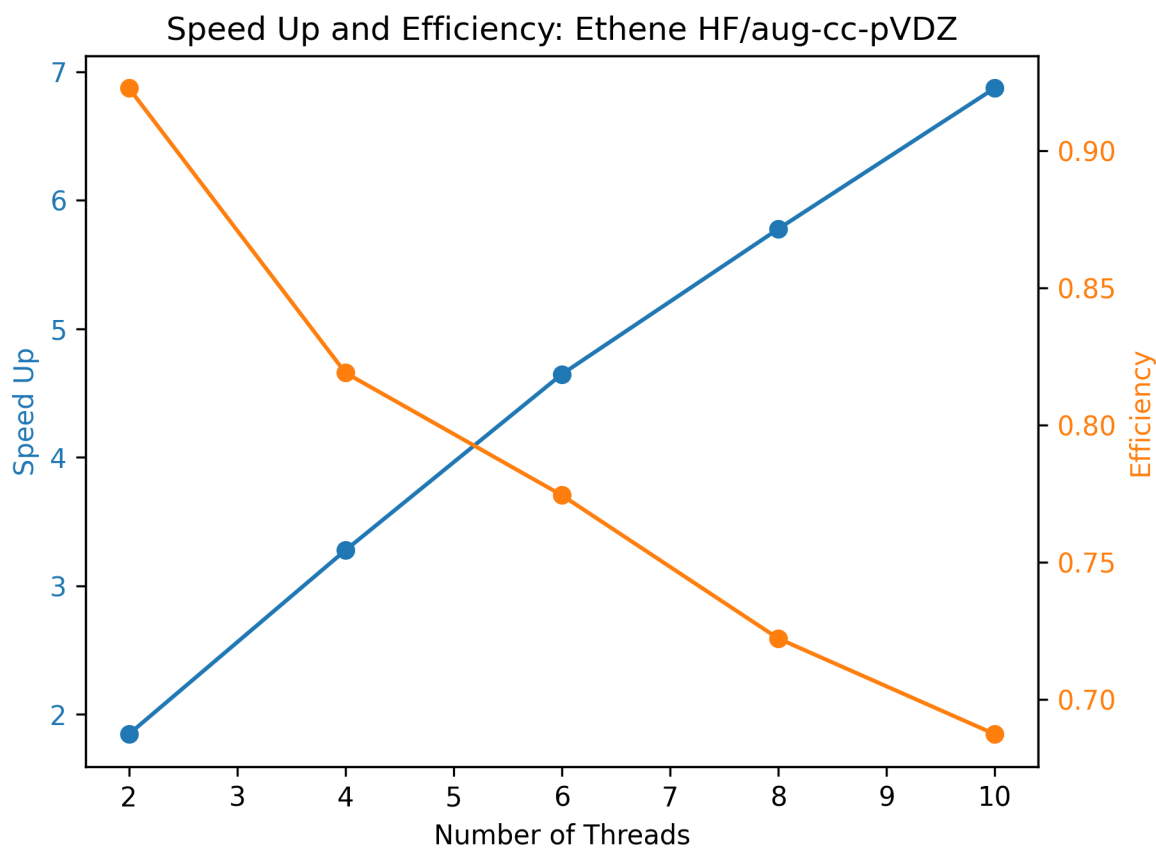
```

```
199
200 Final HF Energy: -78.0435868316762
201
202 Freed Memory
203 Time (CPU) : 221.635326
204 Time (USR) : 28.618671390228
```

Results

The final energy produced from the wather HF/STO-3G yielded -74.965901 Ha while Psi4 on the same geometry and basis set with default options yields -74.965990 Ha. Meanwhile the ethene HF/aug-cc-pVDZ yielded -78.043587 Ha versus the Psi4 result of -78.043510 Ha. The difference between these energies is likely due to me not implementing a cutoff threshold for the commutator of the density and Fock matrix for checking convergence, along with me not using density fitting like Psi4.





The speedups for the water are very marginal and inefficient; however, it is likely due to the size of the system not being large enough and only calling OMP threads for a few of the steps while bottlenecked by serial steps. On the case for ethene with the larger basis set, we experience close to a speedup of 7 at 10 OMP threads, which is still quite inefficient. With that said, the efficiency for only two threads is quite high with nearly a speedup of 2, meaning that adding a few extra threads for this program will give the most improvement for the cost.