```cpp
// GaussianOverlapIntegral.cpp (c) Thursday, October 31, 2024
// by James Pate Williams, Jr., BA, BS, MSWE, PhD
// https://content.wolfram.com/sites/19/2012/02/Ho.pdf

#include <iomanip>
#include <iostream>
using namespace std;
// global molecular data for H20
// position vectors for H20 in yz-plane
// H 1s, O 1s2, 2s2, 2py1, 2pz1
double R[3][3] = {
    { 0.0, 1.43233673, -0.96104039 },
    { 0.0, -1.43233673,-0.96104039 },
    { 0.0, 0.0, 0.24026010 } };
// Gaussian primitive coefficinets
double pCoeff[7][3] = {
    { 0.1543289673, 0.5353281423, 0.4446345422 },
    { 0.1543289673, 0.5353281423, 0.4446345422 },
    { 0.1543289673, 0.5353281423, 0.4446345422 },
    { -0.09996722919, 0.3995128261, 0.7001154689 },
    { 0.155916275, 0.6076837186, 0.3919573931 },
    { 0.155916275, 0.6076837186, 0.3919573931 },
    { 0.155916275, 0.6076837186, 0.3919573931 } };
// Gaussian orbital coefficients
double oCoeff[7][3] = {
    { 3.425250914, 0.6239137298, 0.168855404 },
    { 3.425250914, 0.6239137298, 0.168855404 },
    { 130.7093214, 23.80886605, 6.443608313 },
    { 5.033151319, 1.169596125, 0.38038896 },
    { 5.033151319, 1.169596125, 0.38038896 },
    { 5.033151319, 1.169596125, 0.38038896 },
    { 5.033151319, 1.169596125, 0.38038896 } };
// Cartesian angular momentum quantum numbers
int angMom[7][3] = {
    { 0, 0, 0 }, { 0, 0, 0 },
    { 0, 0, 0 }, { 0, 0, 0 },
    { 1, 0, 0 }, { 0, 1, 0 },
    { 0, 0, 1 } };
// F centers
double FCenter[7][3] = {
    { R[0][0], R[0][1], R[0][2] },
    { R[1][0], R[1][1], R[1][2] },
    { R[2][0], R[2][1], R[2][2] },
    { R[2][0], R[2][1], R[2][2] },
    { R[2][0], R[2][1], R[2][2] },
    { R[2][0], R[2][1], R[2][2] },
    { R[2][0], R[2][1], R[2][2] } };
// iterative n-factorial function
double Factorial(int n)
{
    double factorial = 1;

```

```
53      for (int i = 2; i <= n; i++)
54          factorial *= i;
55
56      return factorial;
57  }
58  // compute the binomial coefficient
59  double Binomial(int n, int k)
60  {
61      return Factorial(n) / (Factorial(k) *
62          Factorial(n - k));
63  }
64  // calculate n!! for even and odd n
65  double DoubleFactorial(int n)
66  {
67      if ((n % 2) == 0)
68      {
69          // even  n
70          double df = 1;
71          for (int i = n; i >= 2; i -= 2)
72              df *= i;
73          return df;
74      }
75      else
76      {
77          // odd n
78          double df = 1;
79          for (int i = n; i >= 3; i -= 2)
80              df *= i;
81          return df;
82      }
83  }
84  // Euclidean norm a.k.a norm-2
85  double Norm2(double A[], double B[])
86  {
87      double ABx = A[0] - B[0];
88      double ABy = A[1] - B[1];
89      double ABz = A[2] - B[2];
90      // compute distance
91      double DAB = sqrt(
92          ABx * ABx + ABy * ABy + ABz * ABz);
93      return DAB;
94  }
95  double EAB(
96      double alpha, double beta,
97      double A[], double B[])
98  {
99      return exp(-alpha * beta * pow(Norm2(A, B), 2.0) /
100         (alpha + beta));
101 }
102 void Pv(
103     double alpha, double beta,
104     double A[], double B[],
```

```
105     double P[])
106 {
107     double denom = alpha + beta;
108     P[0] = (alpha * A[0] + beta * B[0]) / denom;
109     P[1] = (alpha * A[1] + beta * B[1]) / denom;
110     P[2] = (alpha * A[2] + beta * B[2]) / denom;
111 }
112 double Sc(
113     double alpha, double beta,
114     int ax, int bx, double P[],
115     double A[], double B[],
116     int index)
117 {
118     double pi = 4.0 * atan(1.0);
119     double factor = sqrt(pi / (alpha + beta));
120     double sumix = 0;
121     for (int ix = 0; ix <= ax; ix++)
122     {
123         double sumjx = 0;
124         for (int jx = 0; jx <= bx; jx++)
125         {
126             if ((ix + jx) % 2 == 0 )
127             {
128                 double binom1 = Binomial(ax, ix);
129                 double binom2 = Binomial(bx, jx);
130                 double numer1 = DoubleFactorial(ix + jx - 1);
131                 double numer2 = pow(P[index] - A[index], ax - ix);
132                 double numer3 = pow(P[index] - B[index], bx - jx);
133                 double denom1 = pow(2.0 * (alpha + beta),
134                     0.5 * ((double)ix + jx));
135                 sumjx +=
136                     (binom1 * binom2 * numer1 * numer2 *
137                     numer3) / denom1;
138             }
139         }
140         sumix += sumjx;
141     }
142     return factor * sumix;
143 }
144 // overlap integral
145 double S(
146     double alpha, double beta,
147     int ax, int ay, int az,
148     int bx, int by, int bz,
149     double P[], double A[],
150     double B[])
151 {
152     double factor = EAB(alpha, beta, A, B);
153     double Sx = Sc(alpha, beta, ax, bx, P, A, B, 0);
154     double Sy = Sc(alpha, beta, ay, by, P, A, B, 1);
155     double Sz = Sc(alpha, beta, az, bz, P, A, B, 2);
156     return factor * Sx * Sy * Sz;
```

```
157 }
158 // Normalization factor
159 double N(double alpha,
160     int ax, int ay, int az)
161 {
162     double pi = 4.0 * atan(1.0);
163     double factor = pow(2.0 * alpha / pi, 0.75);
164     double numer1 = pow(4.0 * alpha, 0.5 *
165         ((double)ax + ay + az));
166     double denom1 = DoubleFactorial(2 * ax - 1);
167     double denom2 = DoubleFactorial(2 * ay - 1);
168     double denom3 = DoubleFactorial(2 * az - 1);
169     double result = factor * numer1 / sqrt(denom1 *
170         denom2 * denom3);
171     return result;
172 }
173 // overlap between two centers
174 double Smn(int m, int n)
175 {
176     double sumi = 0;
177     for (int i = 0; i < 3; i++)
178     {
179         double sumj = 0;
180         for (int j = 0; j < 3; j++)
181         {
182             double normm = N(
183                 oCoeff[m][i], angMom[m][0],
184                 angMom[m][1], angMom[m][2]);
185             double normn = N(
186                 oCoeff[n][j], angMom[n][0],
187                 angMom[n][1], angMom[n][2]);
188             double pcm = pCoeff[m][i];
189             double pcn = pCoeff[n][j];
190             double fcm[] = {
191                 FCenter[m][0], FCenter[m][1], FCenter[m][2] };
192             double fcn[] = {
193                 FCenter[n][0], FCenter[n][1], FCenter[n][2] };
194             double Pcn[3] = { };
195             Pv(
196                 oCoeff[m][i], oCoeff[n][j],
197                 fcm, fcn, Pcn);
198             double ov = S(
199                 oCoeff[m][i], oCoeff[n][j],
200                 angMom[m][0], angMom[m][1],
201                 angMom[m][2], angMom[n][0],
202                 angMom[n][1], angMom[n][2],
203                 Pcn, fcm, fcn);
204             double term = normm * normn * pcm * pcn * ov;
205             sumj += term;
206         }
207         sumi += sumj;
208     }
```

```
209     return sumi;
210 }
211 int main()
212 {
213     cout << "Overlap Matrix S for H2O" << endl;
214     cout << endl;
215     for (int m = 0; m < 7; m++)
216     {
217         for (int n = 0; n < 7; n++)
218         {
219             double smn = Smn(m, n);
220 
221             if (fabs(smn) > 1.0e-8)
222             {
223                 cout << fixed << setw(9);
224                 cout << smn << "  ";
225             }
226             else
227             {
228                 cout << fixed << setw(9);
229                 cout << 0 << "  ";
230             }
231         }
232         cout << endl;
233     }
234     return 0;
235 }
```