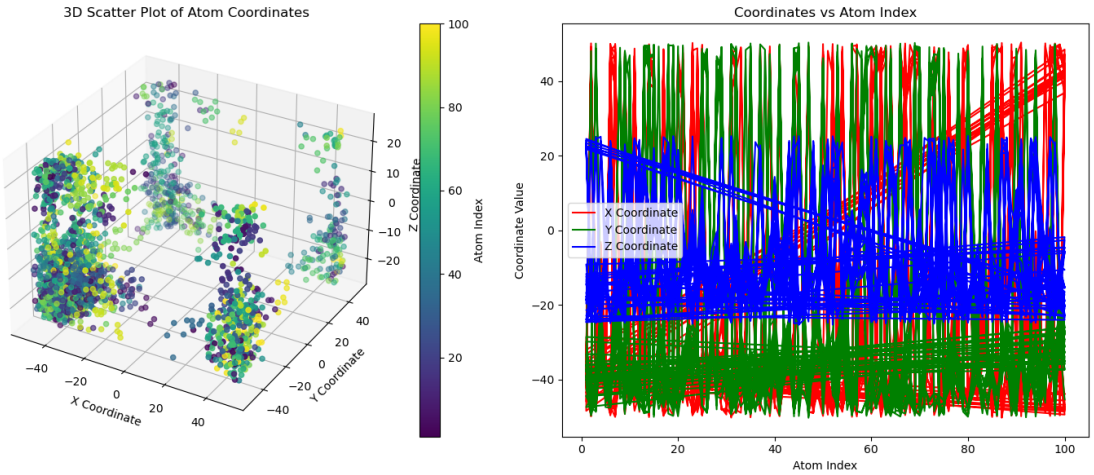
MD Analysis of Water

November 28, 2024

[14]:
```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Load the data from the file
data = np.loadtxt("coord.dat")

# Extract columns: atom index and x, y, z coordinates
atom_index = data[:, 1]
x_coords = data[:, 2]
y_coords = data[:, 3]
z_coords = data[:, 4]

# Create a 3D scatter plot
fig = plt.figure(figsize=(14, 6))

# 3D Scatter Plot of x, y, z coordinates
ax = fig.add_subplot(121, projection='3d')
scatter = ax.scatter(x_coords, y_coords, z_coords, c=atom_index,␣
 ↪cmap='viridis', marker='o')
ax.set_title("3D Scatter Plot of Atom Coordinates")
ax.set_xlabel("X Coordinate")
ax.set_ylabel("Y Coordinate")
ax.set_zlabel("Z Coordinate")
fig.colorbar(scatter, ax=ax, label='Atom Index')

# Line plot of x, y, and z coordinates vs atom index
ax2 = fig.add_subplot(122)
ax2.plot(atom_index, x_coords, label='X Coordinate', color='r')
ax2.plot(atom_index, y_coords, label='Y Coordinate', color='g')
ax2.plot(atom_index, z_coords, label='Z Coordinate', color='b')
ax2.set_title("Coordinates vs Atom Index")
ax2.set_xlabel("Atom Index")
ax2.set_ylabel("Coordinate Value")
ax2.legend()

plt.tight_layout()
```

```
plt.show()
```



[16]:
```
data[:,3]
```

[16]:
```
array([-16.4753685 , -33.63646317,  39.87081528,  31.86639977,
       -13.22975254, -48.21281052,  26.21012688, -38.61767197,
        44.83344269, -39.54844666,  26.14359093, -27.58012009,
       -43.52317429,  43.4822464 , -48.51207733, -19.32457161,
       -45.39871216,  42.95635223,   8.09406853, -44.91402817,
       -48.63732529, -43.36991119, -37.73330688, -28.99528885,
        31.12199402, -41.95713043, -42.711483  , -25.121418  ,
       -35.03873062, -45.42729187,  18.0531826 , -34.67009354,
       -47.96501923,  46.85458374, -38.16675949, -34.30508804,
       -43.14185333, -20.26929855,  44.33857727, -26.41996765,
       -33.20756149, -30.21492767,  47.93689346, -45.66069412,
       -19.37364578, -33.04204559, -36.82344055, -20.9932766 ,
       -14.06022072,  -7.14168596,  47.79251099, -36.4439888 ,
       -22.1280098 , -44.63370514,  38.96903992, -19.02010155,
       -44.90311813,  35.99207306, -41.05653763,  48.30751801,
       -23.05492401, -30.49098587, -25.95401764,  25.79750824,
        25.32266426, -12.93169594, -26.88887596, -43.01706314,
       -47.95246506,  33.35300446, -36.42378235, -39.65162659,
        39.61172867, -21.10824394,  37.15922546, -17.17068481,
       -42.60648346, -42.3325386 , -47.03358841, -40.9127388 ,
        19.79866219,  -4.59879446, -46.34482193,  36.98273468,
       -34.7895546 , -30.32177353, -26.6303215 , -44.59254074,
        -1.04534471,  44.6952858 , -16.38588524,  35.63439941,
       -45.99760818, -31.70653725,  29.15870667, -17.93791008,
       -28.25284195, -16.87668991, -48.71639252, -35.71403885,
       -11.70372868, -34.48400879,  47.40433502,  29.73880577,
```

```
 -4.81947184,  46.61708069,  11.93595886, -37.28091812,
 41.88133621, -26.42696571,  16.82527542, -34.17716599,
 43.72157669, -46.93876648,  46.90482712, -47.83399963,
 39.5310936 , -41.03485107,   3.5962534 ,  44.02633286,
-46.06558609,  43.38864899, -32.40851593, -23.66612816,
 12.873209  , -32.65960693,  41.09952927, -10.61727619,
-37.46114731, -44.54082108,  10.52307796,  48.94629669,
-40.6818924 , -45.78371811, -40.63995743, -39.02153015,
 43.9138031 , -36.54485321,  41.92389679, -17.67934799,
-23.14780998,  48.24970245,  36.29373169,  39.86019135,
-15.53732395, -19.99077988, -28.51978874, -17.05113602,
  1.0553453 ,  -9.81958199, -35.49008179,  38.75997543,
-22.63755417,  43.82987976,  42.01081085,  -2.32046795,
 41.80424118,  27.51101303, -31.70215607, -47.97972107,
 46.07839966, -43.29219437, -23.41954613,  19.07618332,
 21.76846504, -20.78059959, -38.85271072, -24.73322868,
-45.65740967,  24.40821266, -35.5453186 , -37.0233345 ,
 26.11161995, -13.65810108,  47.0724144 , -15.69080639,
 41.97552872, -45.92529678, -41.97302246, -36.885952  ,
 37.31888199, -16.4637394 , -40.93087006,  27.10741234,
-21.62242889, -24.55397606, -23.28959656, -42.00978088,
-19.51268196,  49.18643951,  -9.38178539,  22.40435028,
-42.34863281, -22.85474205,  44.07035065, -22.77742767,
-35.82052231, -44.67202377, -37.29486084, -46.55469894,
  1.5773474 , -35.78788376, -27.14716721,  23.23396301,
 -2.88028789, -39.39273453,  20.94668388, -47.22349167,
 44.1820488 , -31.54482841,  29.59316826, -34.22082138,
 41.94396591, -44.41160202,  44.08056641,  44.85686111,
 39.12106705, -47.23039246,  17.9215374 ,  29.59487915,
 32.11856842,  28.84978867, -41.63644028, -42.54064941,
 11.93722534, -39.23144531, -47.78888321,  -5.07024193,
-48.67237473, -42.24351883,   9.6772995 ,  29.67511749,
-48.42577744, -19.70139313, -29.17124367,  42.61394119,
 38.32395554, -28.29492569,  31.55275154, -20.14632607,
-30.95333862,  28.9900589 ,  40.13187408, -40.51836777,
-23.09117126,   9.34882355, -24.45601654, -23.59415245,
  4.33985138, -18.86153412, -42.1216774 ,  33.58203125,
-27.69235229, -37.17230606, -49.23287964,  10.12717915,
 41.90435791,  27.45207405, -28.77522659,  38.43789673,
 38.30782318,  41.05377197, -36.38474655,   9.98442078,
  9.49607658, -23.83486557, -36.20931244, -14.08255005,
 46.83722687,   6.9595232 , -26.31091118, -26.12495613,
 31.1917057 ,  -5.55617905,  33.97027969, -26.28350258,
 44.63450241,  48.46292114, -23.27443314, -24.35592651,
 42.9428978 , -31.88672066,  47.89321899,   2.67959619,
-26.91745949, -17.61081123, -16.22247124, -38.95812225,
-39.23312378, -46.63424301, -25.50933266,  12.18548679,
```

```
-31.81564903, -24.29081154,  43.06807709, -25.96319962,
-39.92203522, -45.45786285, -30.91884232, -37.17617035,
 39.03568649, -24.46622467, -38.00880051,  27.82787514,
 -2.8126545 , -22.32875252,  23.96433067, -42.80585861,
 28.79301643, -35.94958878,  32.86735916,  45.32110596,
 35.18432236,  40.4352684 ,  32.62629318,  35.54426193,
 35.6882782 ,  47.68389893,  33.25395966,  34.13619232,
 41.03113556,  24.61100006,  46.0753212 , -46.08387375,
 -0.60918343, -20.33665848,  44.90400696, -16.19392014,
 41.93276596, -46.53675079,  27.99840736,  26.20184517,
 48.09634399, -18.85190773, -28.25926208, -37.64598083,
 39.64128494, -33.4800148 ,  38.02508545, -15.72739983,
-24.38582039,  39.4745903 , -44.23826981, -33.66190338,
-26.46597862,  11.38367081, -27.4387989 , -15.33066082,
  6.05843544, -15.81177616, -48.435215  ,  24.08647346,
-21.33338547, -46.89962769, -42.73048401,   5.85437727,
 34.92378616,  33.02084732, -18.44375038,  43.68862152,
 47.43377686,  38.51502991, -35.046772  ,  13.99849796,
 18.92100143, -33.29687119, -39.99560547, -24.98565865,
 48.36021042,  -1.24951351,  -9.69671154, -25.89724731,
 25.63086319,  13.10970116,  17.79114723, -41.96176147,
-43.94510269,  40.95344162, -39.05430222, -31.85509682,
 32.51137543, -28.4751339 ,  31.64016914,  23.61783028,
-28.10551453, -10.36860657,  -8.65147972,  32.79271317,
-37.84530258,  40.2450943 , -31.00857162,   4.97841644,
-31.6566658 , -14.836236  ,  24.36348152, -40.30777359,
-40.68987656, -34.93903351, -38.42160797, -40.24882126,
 37.9127388 , -32.02767944, -37.60620499,  31.21080017,
 -6.05496407, -27.20975494,  24.59179497, -30.62178421,
 35.61055756, -26.84576607,  24.76795387, -45.1097908 ,
 38.4471817 ,  42.92718887,  33.7932663 ,  44.04818726,
 39.12618256, -41.8885231 ,  31.67379189,  18.79225922,
-46.40858459,  21.64548302, -44.99523926,  39.8134613 ,
  0.68045771,   2.05316448,  31.26702881, -25.69539261,
-35.23603439, -35.89319229,  24.81641388,  29.93376923,
 42.91525269, -35.06619263, -11.78667831, -29.45463943,
 29.01034546, -27.79836082,  42.14439774, -11.15056515,
-31.94558716,  30.2011261 ,  42.11814117, -40.09332657,
-10.51268005,  -3.1274507 , -24.28620911, -16.12734222,
 -1.71798611,  -5.70503521, -48.15898514,   7.65094805,
-17.09515381,  44.92733765, -47.74241257,  20.24373055,
 38.16973114,  40.40537643, -48.15769196,  34.22032547,
 48.06695175,  41.77881241, -31.3938446 ,  -2.78850365,
 24.22657967,  48.66008759,  40.99110413, -14.76514435,
-46.13029099,  19.66274071,  10.93013096, -12.15224838,
 21.21641922,   8.38704967,   2.10478258,  36.45503998,
-44.01085281,  25.11338806, -37.72241211, -22.42019653,
```

```
 37.76390839, -25.12426376,  27.28610992,  29.9658432 ,
 -9.8888607 ,   1.79740393,  -4.51351023,  26.62372589,
 45.00388718,  36.41730499, -31.61599159,  -9.4405241 ,
-26.43717384, -23.57107162,  40.44832611, -32.23389435,
-19.82050705, -28.84493828,  47.58639908,  42.20147705,
 35.74703217, -31.06846237, -35.84281921,  28.6119175 ,
-33.79064941, -31.22504807,  17.83407784, -47.70962143,
 35.86516571, -12.34254646,  22.28640938, -45.65615463,
-41.89507294,  18.75797653,  39.35432816,  37.50998306,
 38.3147583 , -46.92774963,  29.26457024,  13.93075848,
 40.81088257,  18.62444687, -25.17412758,  26.17148972,
 -0.84232765, -16.4672966 ,  16.38336563, -15.83956623,
-44.23158264, -48.97118378,  31.21763229,   9.18760204,
 41.40306473, -46.86558914, -14.02981853, -17.39261627,
 25.41115189, -11.31279755,  32.180439  , -15.67747211,
-31.54083633,  32.4275856 ,  43.33537292, -39.46364975,
 -6.28996229, -12.40542793, -31.99514389, -23.35254097,
-10.21623993,  14.2365427 , -40.33595657,  21.04540634,
  9.28417492,  22.95049667,  38.51428223,  32.95541382,
 22.71679306,  36.85067368, -42.13656235,  14.39794445,
 43.87794113, -35.4002037 , -42.19057465,  -8.58869743,
 42.48280716, -41.19604492,  48.43825912,  -9.91839123,
-38.12093353,  17.97646141,   7.57284403,  -5.78037643,
 26.89281082,  25.53762245,  -2.29417872,  19.46155167,
 44.68100357,  20.56519699, -36.6414566 , -29.5805397 ,
-44.52542114, -35.30177307,  23.82921982,  20.92106247,
-17.43246078,  -5.82010031, -11.95542622,  34.69245529,
 39.09082031,  45.25550079, -39.71163177, -13.19464302,
-31.85663986, -17.24136734,  33.04813766, -29.20254707,
-23.65423965, -31.37993622, -47.78455353,  39.99662018,
 46.08735657, -42.98116302, -26.34302521,  16.24017334,
-44.93494797, -29.39222717,  22.34897804,  22.54155731,
 33.52481842,  -4.13360596,  20.66174889, -46.68122482,
-30.39659309,  18.03552246,  18.27372742,  41.87889099,
 39.36181641, -32.20645905,  20.16564178,  20.02115631,
 37.8834877 ,  38.5694809 , -22.41691017,  12.29647923,
  3.13116384,  -7.03174067,  15.31212711, -11.46526337,
-38.58309174, -35.72408295,  22.04422188,  13.94408512,
 43.85001755, -33.50798035, -15.73278141, -17.07575226,
 30.38066864, -32.78079224,  31.00043297, -15.92777348,
-46.60791397,  43.96615601,  29.52687454, -30.72808075,
-23.64535522, -17.09890747, -46.11044693, -20.92235184,
-28.10859108,  20.00164223, -41.41440582,  10.80918217,
 15.3632164 ,  20.9907074 ,  35.32323456,  27.87608147,
 24.88052368, -45.08898163,  34.82292557,  16.51737213,
 37.66808319, -25.74112511, -45.21056366, -14.94530201,
-46.55975342, -31.92250633,  38.24744797, -15.4709177 ,
```

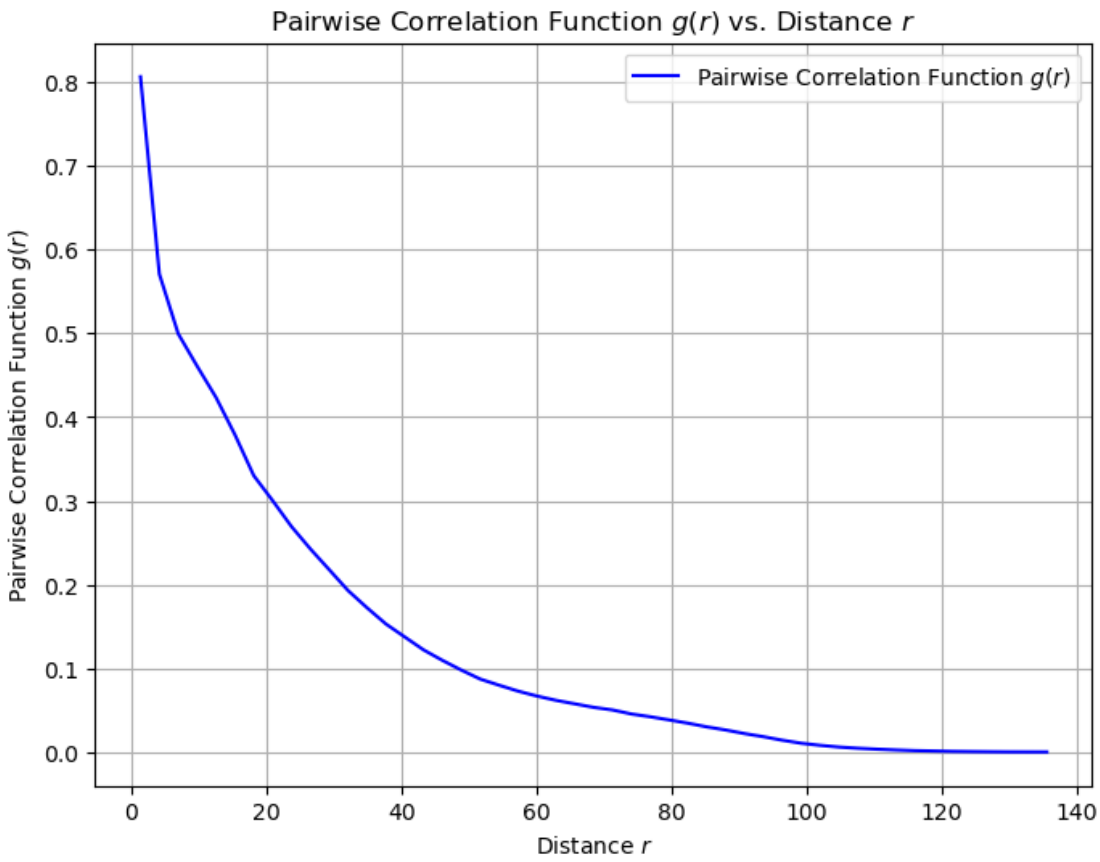
```
-28.31049347,   3.94672966,   1.33053911,  10.25455189,
  7.66300106,  12.14501858,  -3.04707241,  27.18742371,
 42.99518204,  26.86494446, -30.44320107, -23.39425659,
-37.71485901,  48.69587708,  34.15572357,  21.54531097,
-29.42204475,  -4.92369461, -15.95111275, -46.53260422,
 40.32526016,  33.43258667,  46.98618698,  -6.21283102,
-30.48742867, -19.6369133 ,  29.31147957, -37.2339859 ,
-12.4228344 , -39.5141716 ,  39.66910172,  13.46953678,
 43.40290833, -32.21486282, -10.91576385,  36.98340607,
-36.17047501, -32.20010376,  27.39554787,  15.49418449,
-47.57381821,  -0.80857974,  14.68616867, -39.64750671,
-37.76696396,  -3.43263388,  15.55798912,  43.74785614,
 26.65365219, -36.38343048,  29.41972733,  18.42680168,
 29.17132378, -46.88837433, -16.11267853,  10.03076744,
 -2.29650688,   2.41866827,  12.79119682,  -3.34226871,
-42.5690918 , -47.34196472,  -5.66109848,   3.21395731,
 30.48404503, -15.71136284, -12.38049221,  -9.35774136,
 30.75722313, -36.59123611,  18.65472031,  -8.02791309,
-44.542202  , -45.67001343,  31.62375069, -23.45307541,
-24.39127541, -12.87514114, -33.05586243,  -5.27628136,
-42.66759491,  13.69057178, -41.42546463,   8.26949883,
 13.56063557,  21.44197655,  31.86750221,  20.91457176,
 31.44590569, -36.03800964,  29.65996361,  19.5763607 ,
 27.62495232,  -8.84302902, -34.30863953,  -4.71651506,
 36.18305969, -41.76282501,  19.80741692, -26.20920372,
-48.76641464,  17.11158562,  14.64733601,  14.12859917,
 19.03727913,   3.05856252,  19.77672386,  17.22411156,
 48.93524551,  33.65091705, -44.789814  , -19.28771019,
-38.60334396,  48.47275543,  41.52999878,  30.67698479,
-19.91586113,  -5.37415695,  -5.97341347, -31.78593063,
 37.2636528 ,  35.79473114,  40.31833649,  -5.95318508,
-31.11894798, -21.03544617,  38.52262115, -33.14005661,
  3.63871312, -37.18214035, -40.97937393,  10.53590202,
 33.5877037 , -34.80028152,   1.73952174,  37.02744293,
-37.7450943 , -29.63559914, -45.62918472,  11.54111767,
-37.74342728, -13.38132477,   1.26692462, -46.67107391,
-35.80551529, -22.99931145,  13.32391644,  42.15990067,
 29.19363594, -34.098423  ,  26.63894653,  15.03213024,
 29.80988503,  42.1742363 , -13.05104637,   7.68897533,
 13.52689075,  -2.76329207,  31.79411507, -16.28328514,
-29.70209122,  46.9162941 ,   4.82459545,   4.86953259,
 30.77379799, -11.37089825, -13.87287521, -18.19058037,
  1.47970855, -29.88372803,  27.91561317,  -2.98317194,
-37.49848557,  44.95044708,  10.45104027, -24.19271278,
-35.77338028, -10.40776825, -44.16329193, -14.79255581,
-41.14197159,  29.47688866, -39.69205475,  -0.16149624,
 11.69515991,   5.07825756,  42.07839584,  24.70161438,
```

```
 29.8889637 , -46.95189667,  46.64360046,  25.04823494,
-44.67802429, -16.83798409, -33.59869003,  -8.22280025,
 40.25815582,  46.44347763,  23.38532448, -33.18732071,
-44.90104675,  24.69251251,  20.77162933,  13.57227993,
 22.23767281,  -4.6651926 ,   4.57167339,  27.47316933,
 46.2169838 ,  28.92858315,  43.81587601,  -4.91771984,
-45.34236526, -45.04996872,  30.00602913,  35.39133453,
-20.62698364, -10.25684166, -13.25915718, -45.39608002,
 43.45018768,  40.24790192,  45.49609375,  -2.41531968,
-22.05627632, -25.021698  ,  36.55200958, -35.26443481,
-12.35267735, -39.44433975, -41.28755188,  19.58630562,
 12.17379951, -28.93659592,  -8.9286623 ,  29.39492035,
-45.86605835, -34.50365448, -26.02847099,   3.32593393,
-48.36497116, -18.88809395,   8.09377766, -43.80718994,
-42.02590561, -27.3808918 ,  10.41035366, -45.11741638,
 18.31533432, -38.03171539,  18.54714584,  32.75421906,
 34.86667252,  33.18779755,  -7.26320362,  -0.95205063,
 17.80409622,   6.40663624,  30.21066856,  -3.06709456,
-35.34573746, -32.65322876,   6.77340221,  -0.66314071,
 25.66621208,  -6.08157206,  11.91265106, -14.87376499,
-25.03583717, -29.77513885,  28.03420067,  -3.5137639 ,
-36.2095871 , -48.2450943 ,  22.89597321, -29.74033928,
-28.62538147,  -3.43068004, -32.74881744, -27.22760773,
-37.40421677, -49.17131424, -39.52461624,  -1.12124777,
 15.7261219 ,  24.71735191, -44.38589478,  38.93843079,
 20.3250103 , -31.10656548, -35.76094055,  24.86991692,
-41.30158234, -19.06883621, -41.68873978, -10.01801109,
-43.65738297, -46.93026733,  39.85692978, -36.50102615,
 41.98857117,  24.24422646,  23.98579025,   3.20316195,
  8.0688591 , -16.04276085,  20.99081993,  30.72402   ,
 43.69182587,  32.49417114,  45.70705795,   0.07713456,
 38.52261353, -46.23662567,  14.24838161,  26.56651878,
-26.55814934,  -9.17517567, -18.2650547 , -47.67946625,
 42.46113968,  34.16860199,  30.0935421 ,  -9.03335667,
-33.2089653 , -32.27050018,  48.98314285, -28.62875175,
-21.0300312 , -30.46970177, -43.59887695,  29.06296349])
```

[ ]:

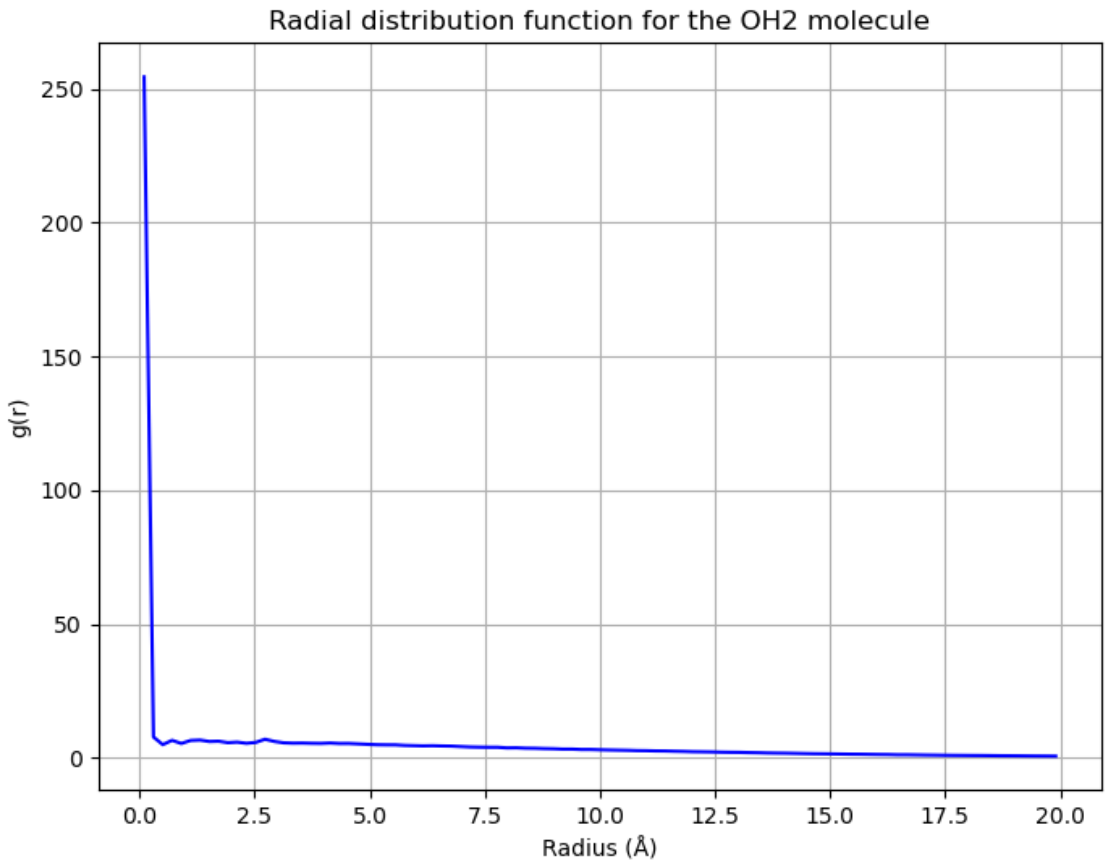
[8]:
```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.spatial.distance import pdist

# Load the data
data = np.loadtxt('coord1.dat')
x_coords = data[:, 2]
y_coords = data[:, 3]
```

```
z_coords = data[:, 4]

# Number of atoms and density estimation
num_atoms = len(x_coords)
density = num_atoms / ((np.ptp(x_coords) * np.ptp(y_coords) * np.
 ↪ptp(z_coords)))  # estimate density

# Calculate pairwise distances
pairwise_distances = pdist(np.column_stack((x_coords, y_coords, z_coords)))

# Define bins for the histogram
bin_edges = np.linspace(0, np.max(pairwise_distances), 50)
dr = bin_edges[1] - bin_edges[0]

# Compute histogram (count of pairs within each spherical shell)
hist, bin_edges = np.histogram(pairwise_distances, bins=bin_edges)
r_values = 0.5 * (bin_edges[1:] + bin_edges[:-1])

# Volume of each spherical shell
shell_volumes = 4 * np.pi * (r_values**2) * dr

# Calculate g(r)
g_r = hist / (shell_volumes * density * num_atoms)

# Plot g(r) vs. r
plt.figure(figsize=(8, 6))
plt.plot(r_values, g_r, label="Pairwise Correlation Function $g(r)$", color='b')
plt.xlabel("Distance $r$")
plt.ylabel("Pairwise Correlation Function $g(r)$")
plt.title("Pairwise Correlation Function $g(r)$ vs. Distance $r$")
plt.legend()
plt.grid()
plt.show()
```



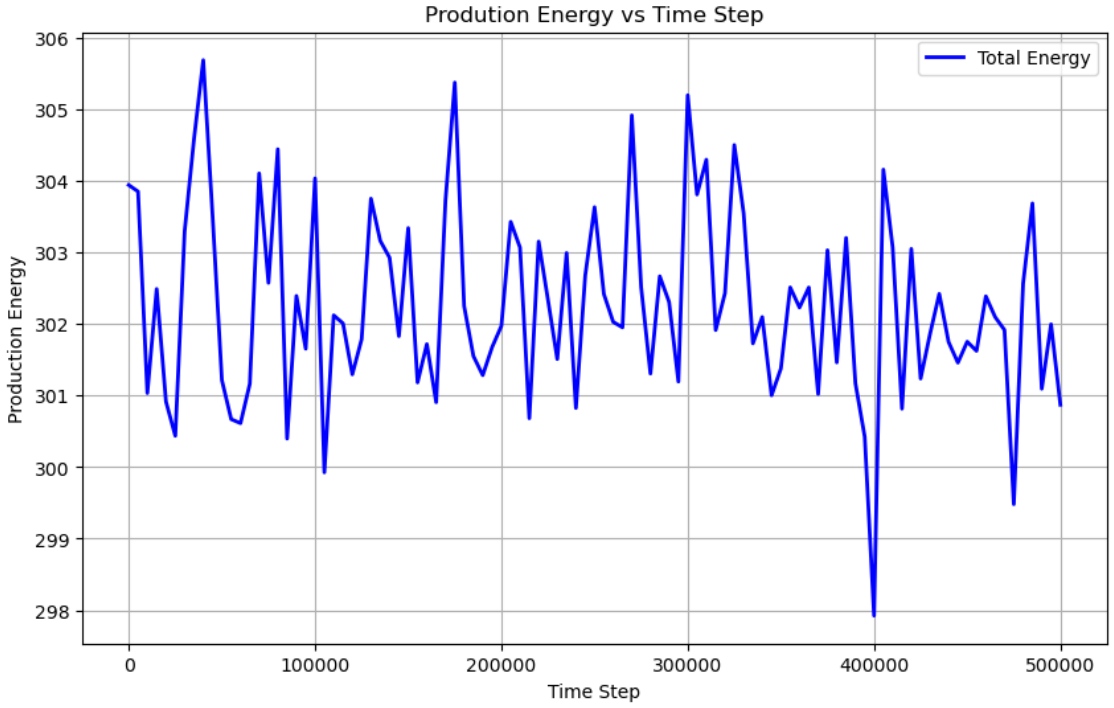
[7]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.spatial.distance import pdist

# Load the data
data = np.loadtxt('coord.dat')
x_coords = data[:, 2]
y_coords = data[:, 3]
z_coords = data[:, 4]

# Number of atoms and density estimation
num_atoms = len(x_coords)
density = num_atoms / ((np.ptp(x_coords) * np.ptp(y_coords) * np.
 ↪ptp(z_coords)))  # estimate density

# Calculate pairwise distances
pairwise_distances = pdist(np.column_stack((x_coords, y_coords, z_coords)))
```

```
# Define bin edges to match the desired range up to 20 Å with appropriate bin␣
 ↪width
bin_edges = np.linspace(0, 20, 100)  # 100 bins for higher resolution within 20␣
 ↪Å
dr = bin_edges[1] - bin_edges[0]

# Compute histogram (count of pairs within each spherical shell)
hist, bin_edges = np.histogram(pairwise_distances, bins=bin_edges)
r_values = 0.5 * (bin_edges[1:] + bin_edges[:-1])

# Volume of each spherical shell
shell_volumes = 4 * np.pi * (r_values**2) * dr

# Calculate g(r)
g_r = hist / (shell_volumes * density * num_atoms)

# Plot g(r) vs. r to match the target style
plt.figure(figsize=(8, 6))
plt.plot(r_values, g_r, label="Radial distribution function for the OH2␣
 ↪molecule", color='b')
plt.xlabel("Radius (Å)")
plt.ylabel("g(r)")
plt.title("Radial distribution function for the OH2 molecule")
plt.grid(True)
plt.show()
```



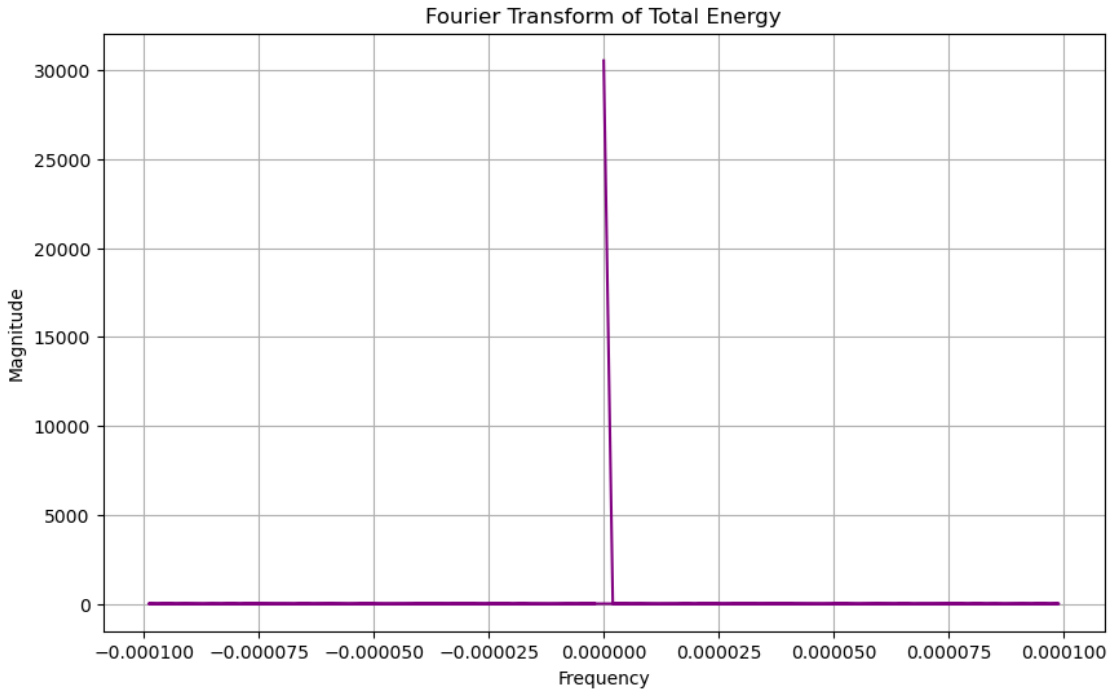
[6]:
```
import numpy as np
import matplotlib.pyplot as plt

# Define a function to read and clean the data
def load_energy_data(filename):
    # Open the file and parse each line
    data = []
    with open(filename, 'r') as file:
        for line in file:
            if line.startswith("ENERGY:"):
                # Split the line, ignore "ENERGY:", and convert the rest to␣
 ↪floats
                values = line.split()[1:]
                data.append([float(val) for val in values])
    return np.array(data)

# Load the cleaned data
data = load_energy_data('prod_ener.txt')
```

```
# Extract the columns for Time Step and Total Energy
time_step = data[:, 0]     # Column 1 is Time Step
total_energy = data[:, 11]  # Column 12 is Total Energy

# Plotting
plt.figure(figsize=(10, 6))
plt.plot(time_step, total_energy, label='Total Energy', color='blue', lw=2)
plt.xlabel('Time Step')
plt.ylabel('Production Energy')
plt.title('Prodution Energy vs Time Step')
plt.grid(True)
plt.legend()
plt.savefig('energy_plot.png')  # Save plot as PNG file
plt.show()
```



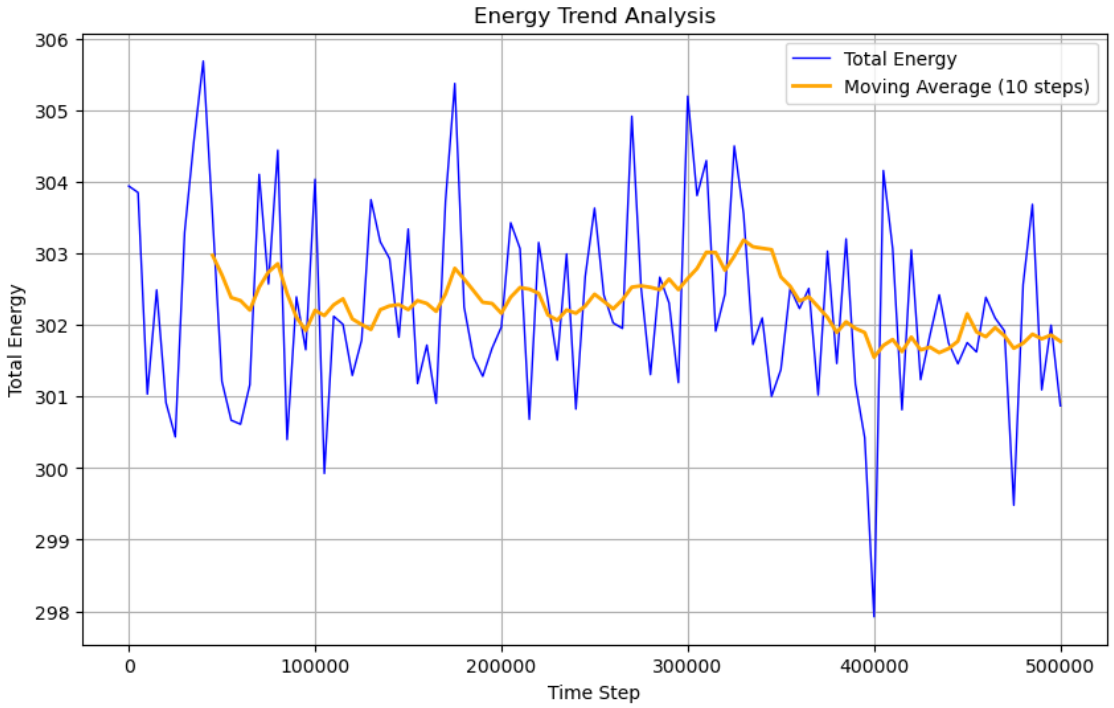
[7]:
```
import numpy as np
import matplotlib.pyplot as plt

# Define a function to read and clean the data
def load_energy_data(filename):
    data = []
    with open(filename, 'r') as file:
        for line in file:
            if line.startswith("ENERGY:"):
```

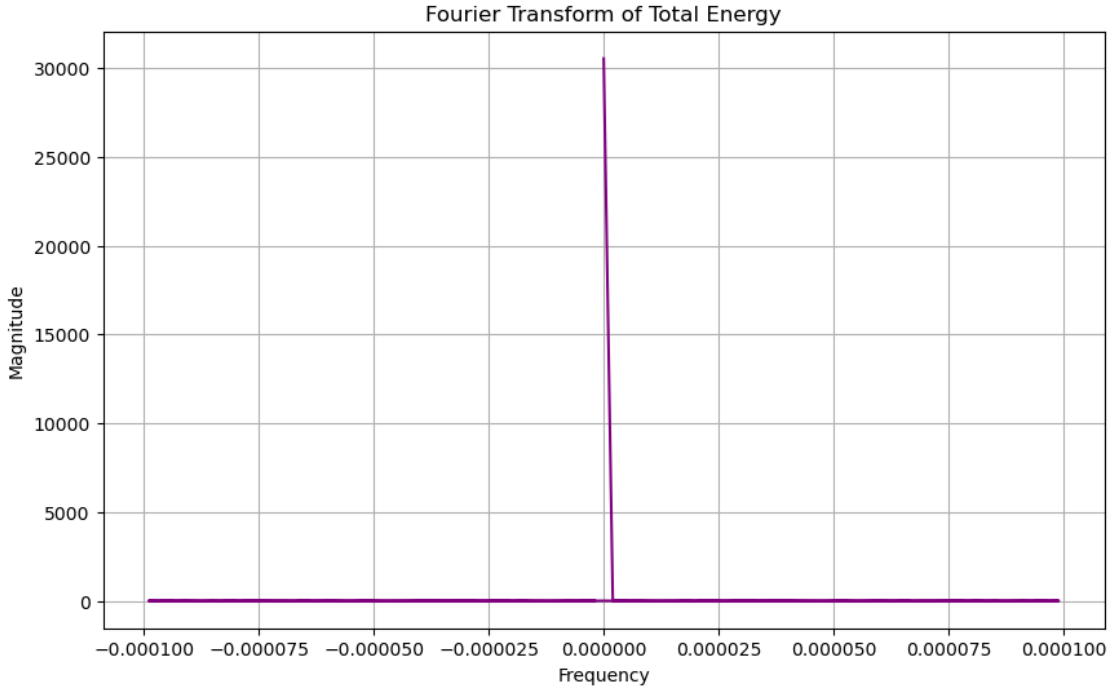
```
                values = line.split()[1:]
                data.append([float(val) for val in values])
    return np.array(data)

# Load the data
data = load_energy_data('prod_ener.txt')

# Extract time step and total energy
time_step = data[:, 0]
total_energy = data[:, 11]

# Perform FFT on the total energy
energy_fft = np.fft.fft(total_energy)
frequencies = np.fft.fftfreq(len(total_energy), d=(time_step[1] - time_step[0]))

# Plot the magnitude of the Fourier transform
plt.figure(figsize=(10, 6))
plt.plot(frequencies, np.abs(energy_fft), color='purple')
plt.xlabel('Frequency')
plt.ylabel('Magnitude')
plt.title('Fourier Transform of Total Energy')
plt.grid(True)
plt.show()
```

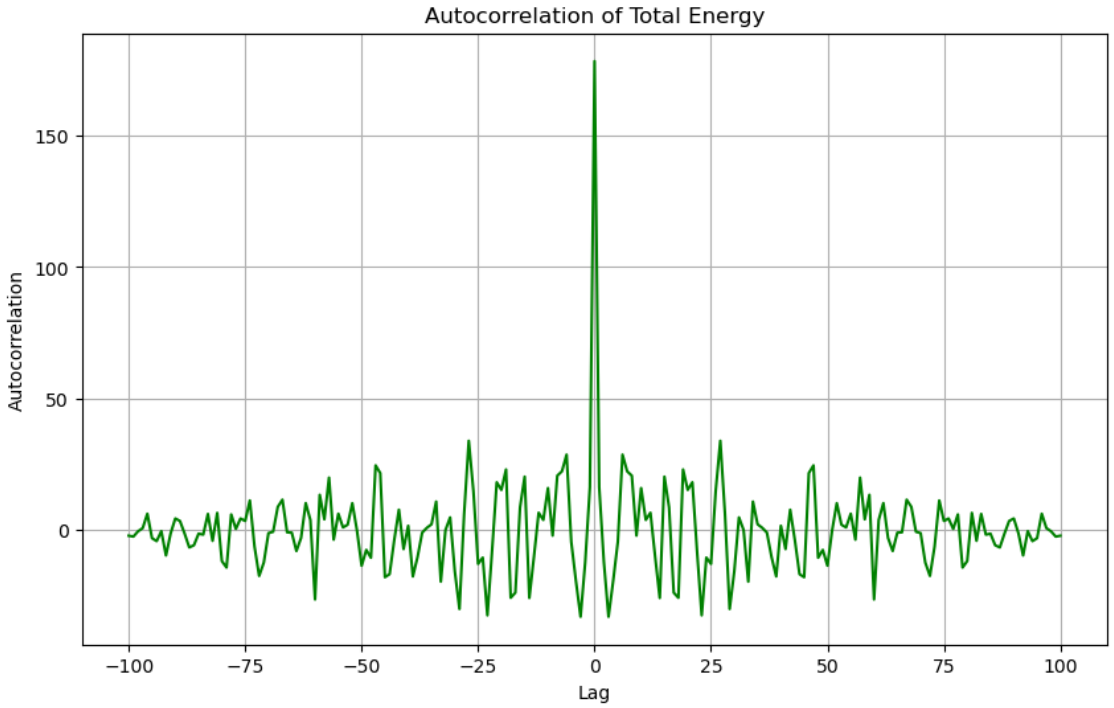


[13]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.fft import fft, fftfreq
from scipy.signal import correlate

# Load data function
def load_energy_data(filename):
    data = []
    with open(filename, 'r') as file:
        for line in file:
            if line.startswith("ENERGY:"):
                values = line.split()[1:]
                data.append([float(val) for val in values])
    return np.array(data)

# Load and extract relevant data
data = load_energy_data('prod_ener.txt')
time_step = data[:, 0]
total_energy = data[:, 11]

# 1. Energy Trend Analysis
mean_energy = np.mean(total_energy)
variance_energy = np.var(total_energy)

# Moving Average (window size of 10 steps for example)
window_size = 10
moving_avg_energy = np.convolve(total_energy, np.ones(window_size)/window_size,␣
 ↪mode='valid')

plt.figure(figsize=(10, 6))
plt.plot(time_step, total_energy, label='Total Energy', color='blue', lw=1)
plt.plot(time_step[window_size-1:], moving_avg_energy, label='Moving Average␣
 ↪(10 steps)', color='orange', lw=2)
plt.xlabel('Time Step')
plt.ylabel('Total Energy')
plt.title('Energy Trend Analysis')
plt.grid(True)
plt.legend()
plt.show()

# 2. Frequency Analysis (Fourier Transform)
energy_fft = fft(total_energy)
frequencies = fftfreq(len(total_energy), d=(time_step[1] - time_step[0]))

plt.figure(figsize=(10, 6))
plt.plot(frequencies, np.abs(energy_fft), color='purple')
plt.xlabel('Frequency')
```

```
plt.ylabel('Magnitude')
plt.title('Fourier Transform of Total Energy')
plt.grid(True)
plt.show()

# 3. Autocorrelation Analysis
autocorrelation = correlate(total_energy - mean_energy, total_energy - mean_energy, mode='full')
lags = np.arange(-len(total_energy) + 1, len(total_energy))

plt.figure(figsize=(10, 6))
plt.plot(lags, autocorrelation, color='green')
plt.xlabel('Lag')
plt.ylabel('Autocorrelation')
plt.title('Autocorrelation of Total Energy')
plt.grid(True)
plt.show()

# 4. Bond Energy and Bond Strength Estimation (Simplified)
# Estimate fluctuations as a proxy for bond stability
fluctuations = total_energy - mean_energy
bond_energy = np.mean(np.abs(fluctuations))
bond_strength = np.max(np.abs(fluctuations)) - np.min(np.abs(fluctuations))

print(f"Mean Total Energy: {mean_energy}")
print(f"Variance of Total Energy: {variance_energy}")
print(f"Approximate Bond Energy: {bond_energy}")
print(f"Approximate Bond Strength (Range of Fluctuations): {bond_strength}")
```


Energy Trend Analysis
Total Energy
Moving Average (10 steps)
Total Energy
Time Step
306
305
304
303
302
301
300
299
298
0
100000
200000
300000
400000
500000


Fourier Transform of Total Energy
Magnitude
Frequency
30000
25000
20000
15000
10000
5000
0
−0.000100
−0.000075
−0.000050
−0.000025
0.000000
0.000025
0.000050
0.000075
0.000100




```
Mean Total Energy: 302.27101881188116
Variance of Total Energy: 1.7672740203391823
Approximate Bond Energy: 1.0520537790412687
Approximate Bond Strength (Range of Fluctuations): 4.3308999999999855
```

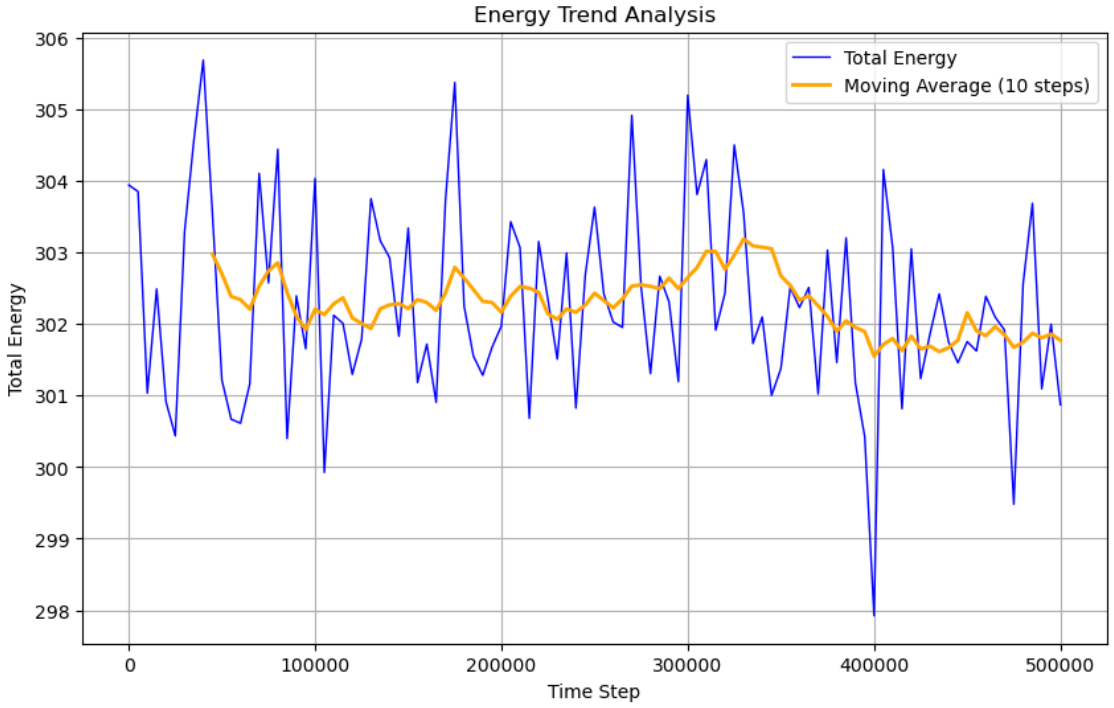
[8]:
```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.fft import fft, fftfreq
from scipy.signal import correlate

# Load data function
def load_energy_data(filename):
    data = []
    with open(filename, 'r') as file:
        for line in file:
            if line.startswith("ENERGY:"):
                values = line.split()[1:]
                data.append([float(val) for val in values])
    return np.array(data)

# Load and extract relevant data
data = load_energy_data('prod_ener.txt')
time_step = data[:, 0]
total_energy = data[:, 11]
```

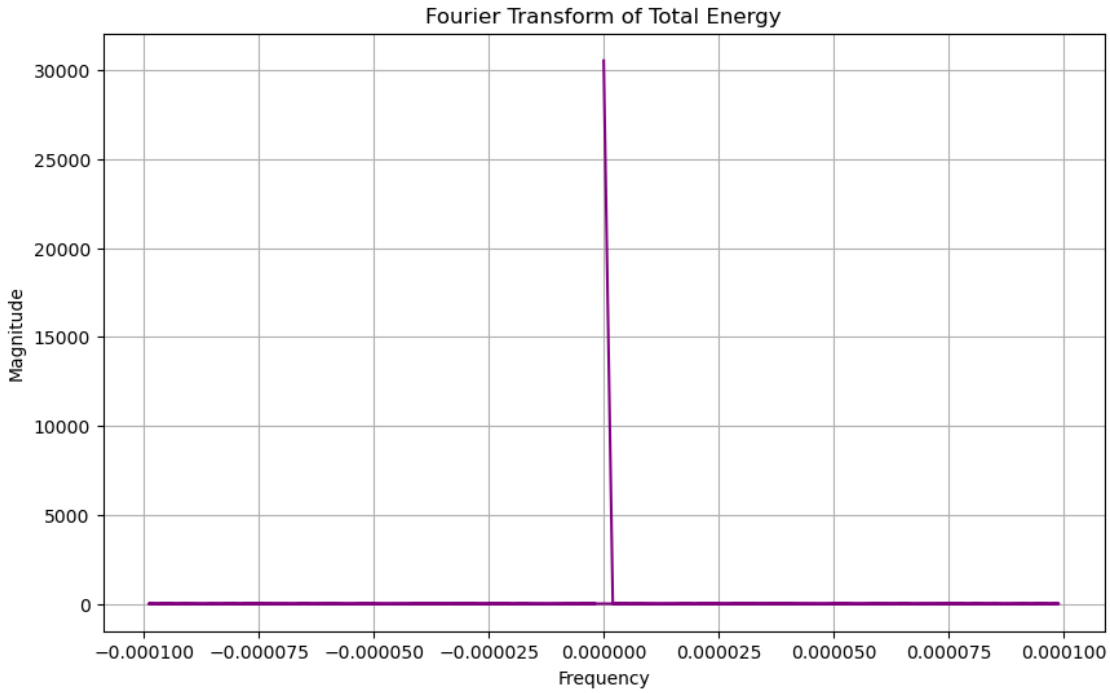
```
# 1. Energy Trend Analysis
mean_energy = np.mean(total_energy)
variance_energy = np.var(total_energy)

# Moving Average (window size of 10 steps for example)
window_size = 10
moving_avg_energy = np.convolve(total_energy, np.ones(window_size)/window_size,␣
 ↪mode='valid')

plt.figure(figsize=(10, 6))
plt.plot(time_step, total_energy, label='Total Energy', color='blue', lw=1)
plt.plot(time_step[window_size-1:], moving_avg_energy, label='Moving Average␣
 ↪(10 steps)', color='orange', lw=2)
plt.xlabel('Time Step')
plt.ylabel('Total Energy')
plt.title('Energy Trend Analysis')
plt.grid(True)
plt.legend()
plt.show()

# 2. Frequency Analysis (Fourier Transform)
energy_fft = fft(total_energy)
frequencies = fftfreq(len(total_energy), d=(time_step[1] - time_step[0]))

plt.figure(figsize=(10, 6))
plt.plot(frequencies, np.abs(energy_fft), color='purple')
plt.xlabel('Frequency')
plt.ylabel('Magnitude')
plt.title('Fourier Transform of Total Energy')
plt.grid(True)
plt.show()

# 3. Autocorrelation Analysis
autocorrelation = correlate(total_energy - mean_energy, total_energy -␣
 ↪mean_energy, mode='full')
lags = np.arange(-len(total_energy) + 1, len(total_energy))

plt.figure(figsize=(10, 6))
plt.plot(lags, autocorrelation, color='green')
plt.xlabel('Lag')
plt.ylabel('Autocorrelation')
plt.title('Autocorrelation of Total Energy')
plt.grid(True)
plt.show()

# 4. Bond Energy and Bond Strength Estimation (Simplified)
```

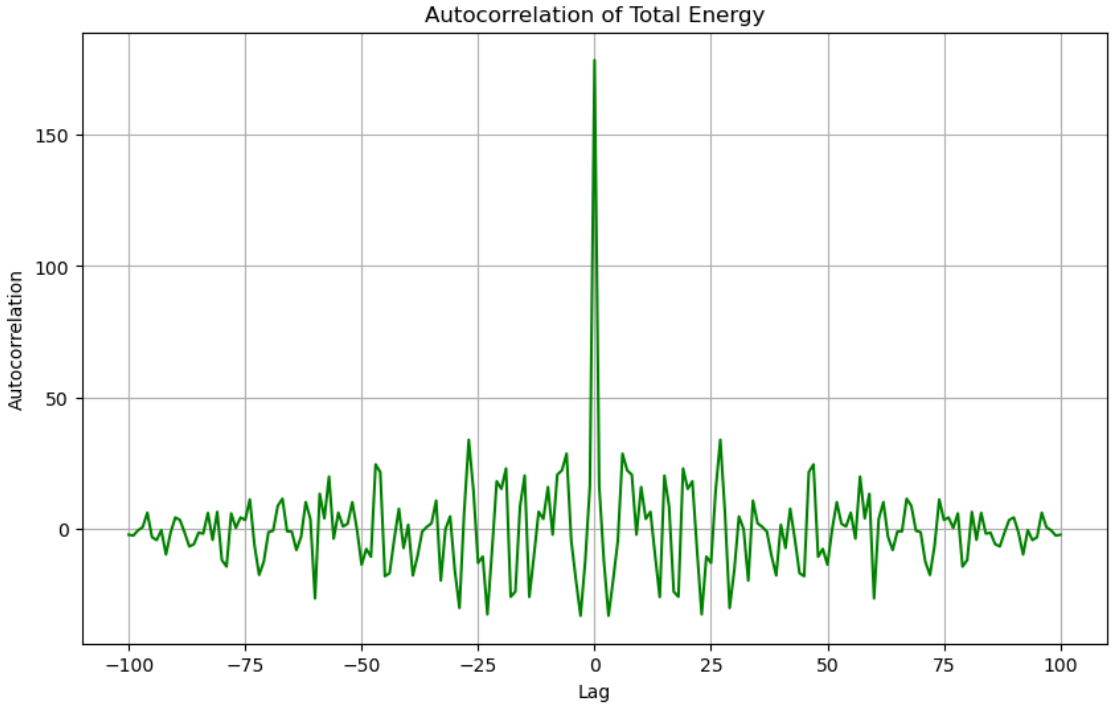
```
# Estimate fluctuations as a proxy for bond stability
fluctuations = total_energy - mean_energy

# Average fluctuation in total energy (in Joules)
bond_energy = np.mean(np.abs(fluctuations))

# Convert the bond energy to kJ/mol using Avogadro's number and scaling
NA = 6.022e23  # Avogadro's number (mol^-1)
bond_energy_kJmol = bond_energy * NA * 1e-3  # Convert to kJ/mol

# Bond strength (range of fluctuations)
bond_strength = np.max(np.abs(fluctuations)) - np.min(np.abs(fluctuations))

print(f"Mean Total Energy: {mean_energy}")
print(f"Variance of Total Energy: {variance_energy}")
print(f"Approximate Bond Energy (kJ/mol): {bond_energy_kJmol}")
print(f"Approximate Bond Strength (Range of Fluctuations): {bond_strength}")
import numpy as np
import matplotlib.pyplot as plt
from scipy.fft import fft, fftfreq
from scipy.signal import correlate

# Load data function
def load_energy_data(filename):
    data = []
    with open(filename, 'r') as file:
        for line in file:
            if line.startswith("ENERGY:"):
                values = line.split()[1:]
                data.append([float(val) for val in values])
    return np.array(data)

# Load and extract relevant data
data = load_energy_data('prod_ener.txt')
time_step = data[:, 0]
total_energy = data[:, 11]

# 1. Energy Trend Analysis
mean_energy = np.mean(total_energy)
variance_energy = np.var(total_energy)

# Moving Average (window size of 10 steps for example)
window_size = 10
moving_avg_energy = np.convolve(total_energy, np.ones(window_size)/window_size,␣
 ↪mode='valid')

plt.figure(figsize=(10, 6))
```

```
plt.plot(time_step, total_energy, label='Total Energy', color='blue', lw=1)
plt.plot(time_step[window_size-1:], moving_avg_energy, label='Moving Average␣
 ↪(10 steps)', color='orange', lw=2)
plt.xlabel('Time Step')
plt.ylabel('Total Energy')
plt.title('Energy Trend Analysis')
plt.grid(True)
plt.legend()
plt.show()

# 2. Frequency Analysis (Fourier Transform)
energy_fft = fft(total_energy)
frequencies = fftfreq(len(total_energy), d=(time_step[1] - time_step[0]))

plt.figure(figsize=(10, 6))
plt.plot(frequencies, np.abs(energy_fft), color='purple')
plt.xlabel('Frequency')
plt.ylabel('Magnitude')
plt.title('Fourier Transform of Total Energy')
plt.grid(True)
plt.show()

# 3. Autocorrelation Analysis
autocorrelation = correlate(total_energy - mean_energy, total_energy -␣
 ↪mean_energy, mode='full')
lags = np.arange(-len(total_energy) + 1, len(total_energy))

plt.figure(figsize=(10, 6))
plt.plot(lags, autocorrelation, color='green')
plt.xlabel('Lag')
plt.ylabel('Autocorrelation')
plt.title('Autocorrelation of Total Energy')
plt.grid(True)
plt.show()

# 4. Bond Energy and Bond Strength Estimation (Simplified)
# Estimate fluctuations as a proxy for bond stability
fluctuations = total_energy - mean_energy

# Average fluctuation in total energy (in Joules)
bond_energy = np.mean(np.abs(fluctuations))

# Convert the bond energy to kJ/mol using Avogadro's number and scaling
NA = 6.022e23  # Avogadro's number (mol^-1)
bond_energy_kJmol = bond_energy * NA * 1e-3  # Convert to kJ/mol

# Bond strength (range of fluctuations)
```

```
bond_strength = np.max(np.abs(fluctuations)) - np.min(np.abs(fluctuations))

print(f"Mean Total Energy: {mean_energy}")
print(f"Variance of Total Energy: {variance_energy}")
print(f"Approximate Bond Energy (kJ/mol): {bond_energy_kJmol}")
print(f"Approximate Bond Strength (Range of Fluctuations): {bond_strength}")
```

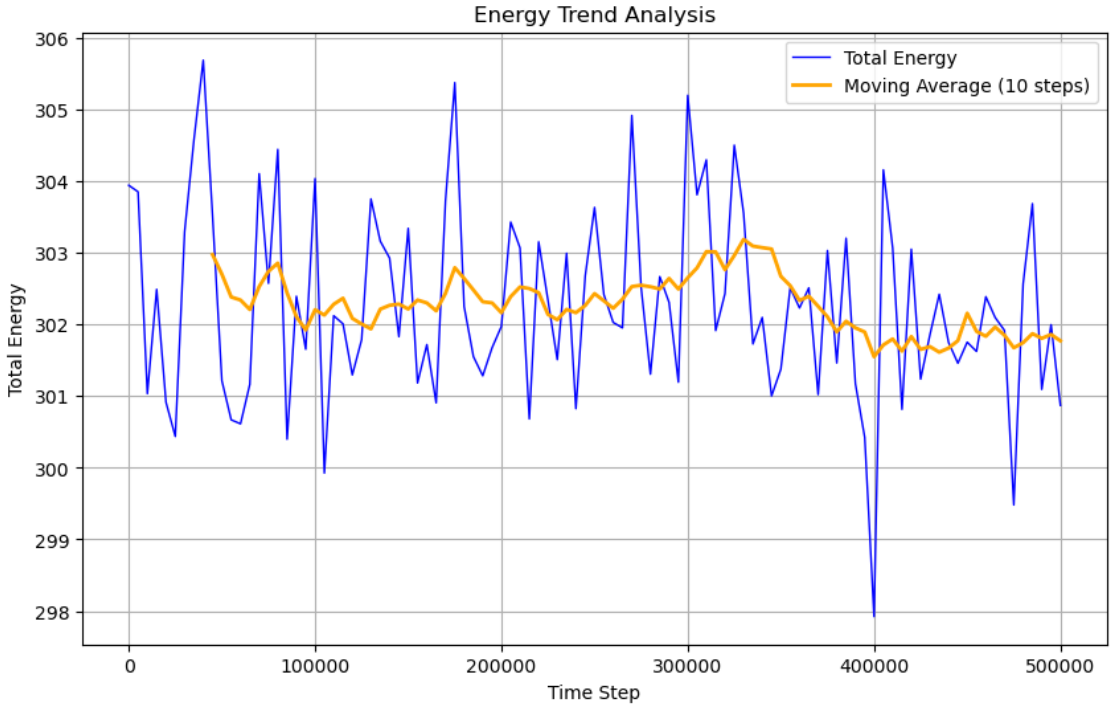


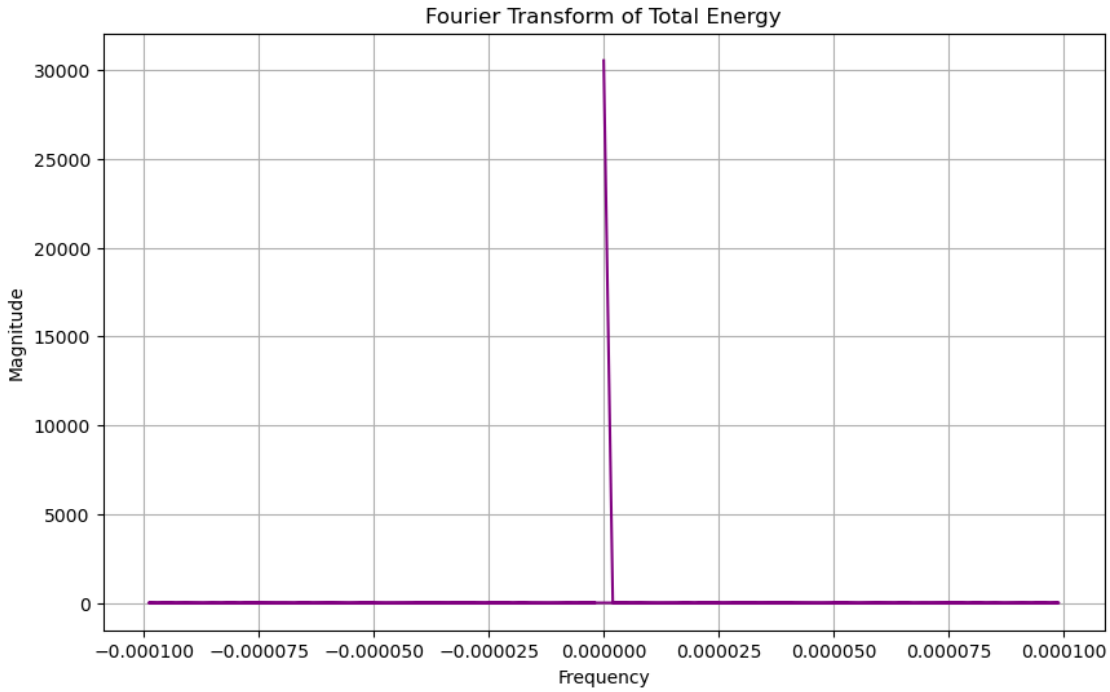


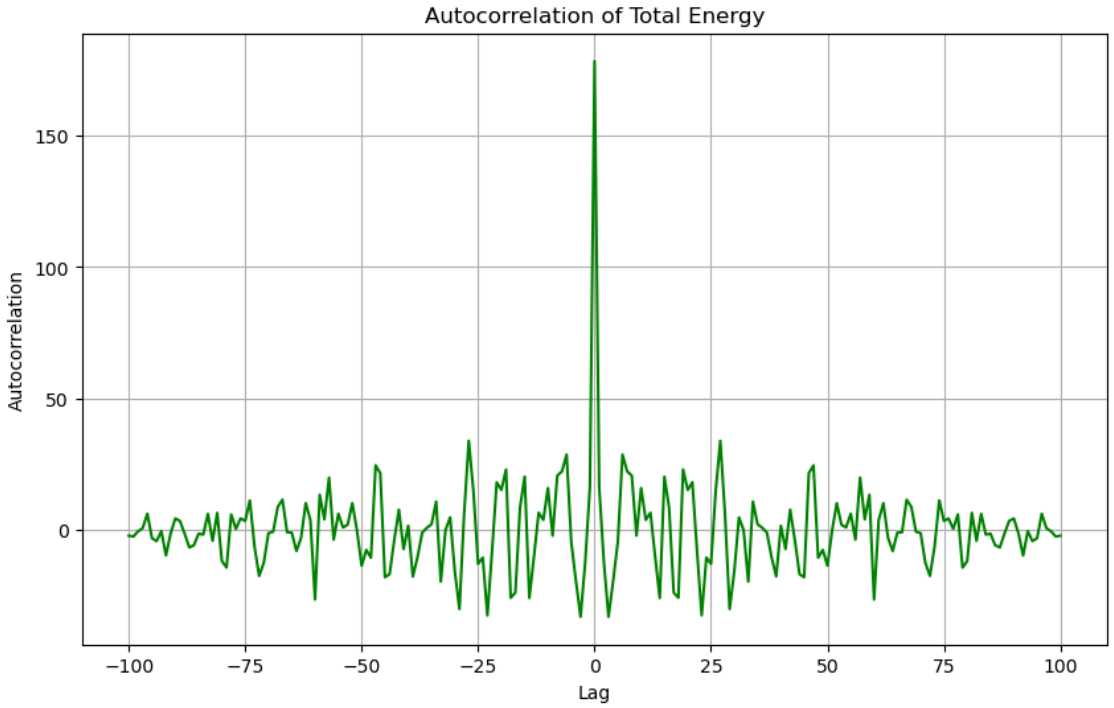


Mean Total Energy: 302.27101881188116
Variance of Total Energy: 1.7672740203391823

```
Approximate Bond Energy (kJ/mol): 6.335467857386521e+20
Approximate Bond Strength (Range of Fluctuations): 4.3308999999999855
```





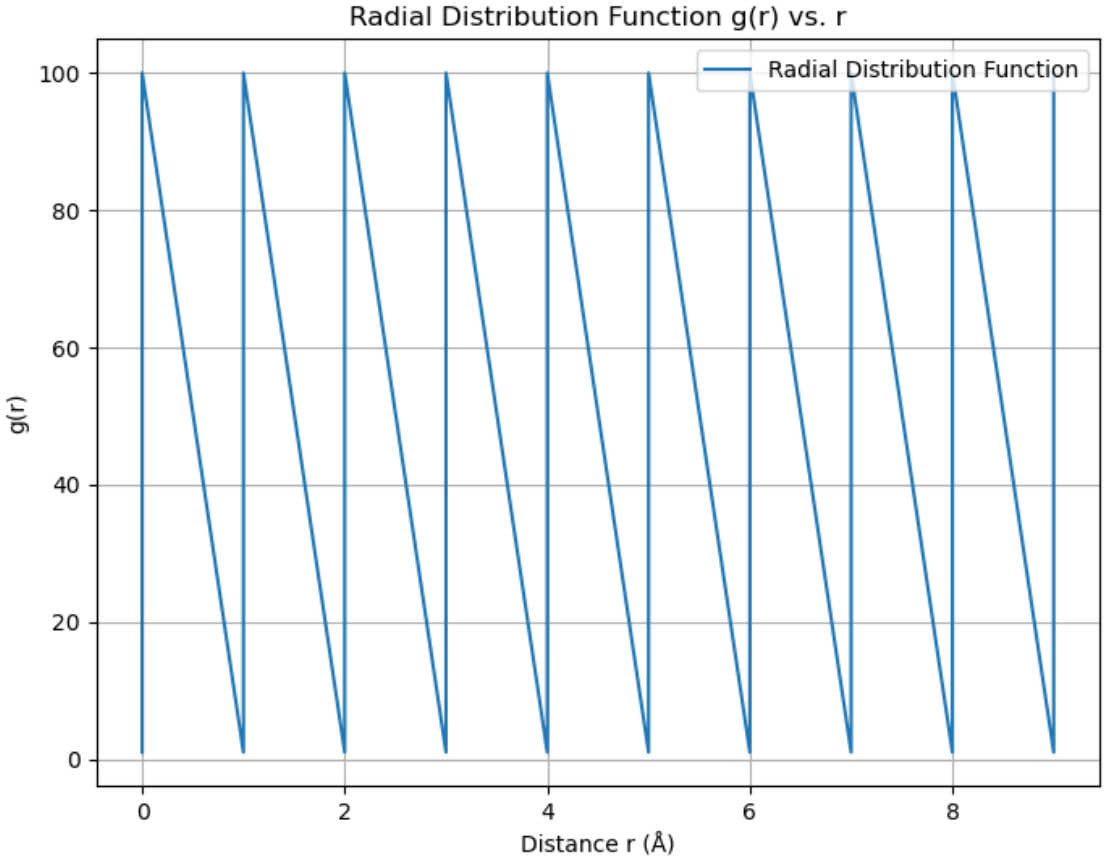



```
Mean Total Energy: 302.27101881188116
Variance of Total Energy: 1.7672740203391823
Approximate Bond Energy (kJ/mol): 6.335467857386521e+20
Approximate Bond Strength (Range of Fluctuations): 4.3308999999999855
```

[9]:
```python
# Convert from kcal/mol to kJ/mol
kcal_to_kJmol = 4.184  # 1 kcal/mol = 4.184 kJ/mol

# Estimate fluctuations as a proxy for bond stability in kcal/mol
fluctuations_kcal = total_energy - mean_energy
bond_energy_kcal = np.mean(np.abs(fluctuations_kcal))

# Convert bond energy to kJ/mol
bond_energy_kJmol = bond_energy_kcal * kcal_to_kJmol

print(f"Mean Total Energy: {mean_energy}")
print(f"Variance of Total Energy: {variance_energy}")
print(f"Approximate Bond Energy (kJ/mol): {bond_energy_kJmol}")
print(f"Approximate Bond Strength (Range of Fluctuations): {bond_strength}")
```

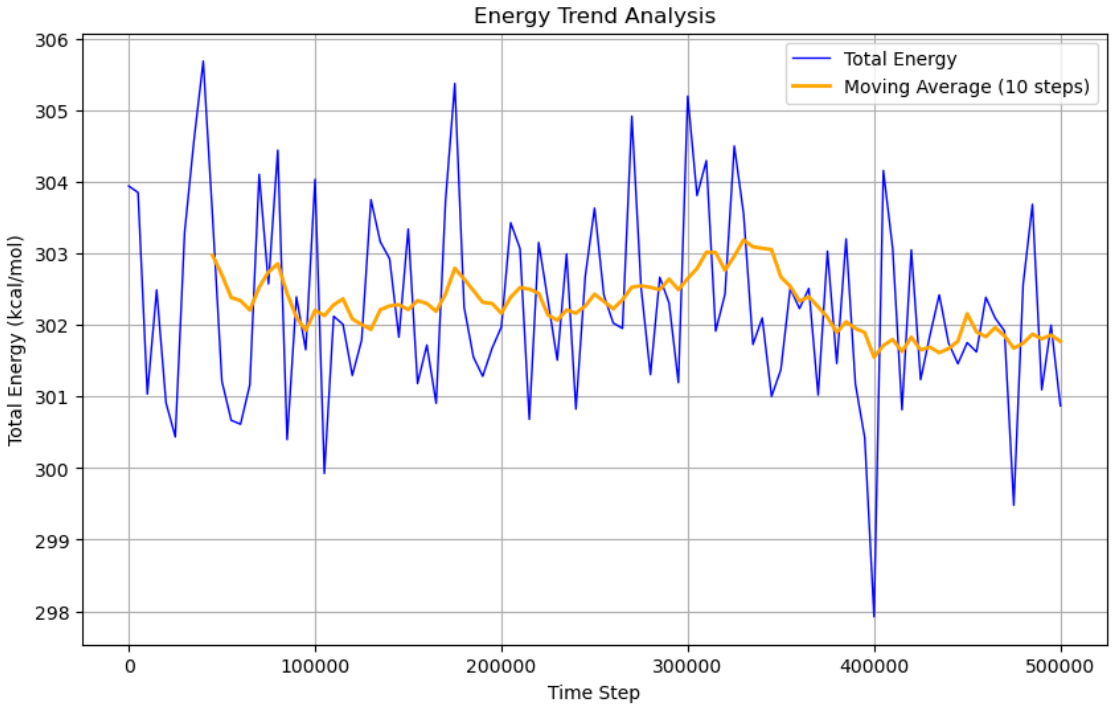
```
Mean Total Energy: 302.27101881188116
Variance of Total Energy: 1.7672740203391823
Approximate Bond Energy (kJ/mol): 4.401793011508668
Approximate Bond Strength (Range of Fluctuations): 4.3308999999999855
```

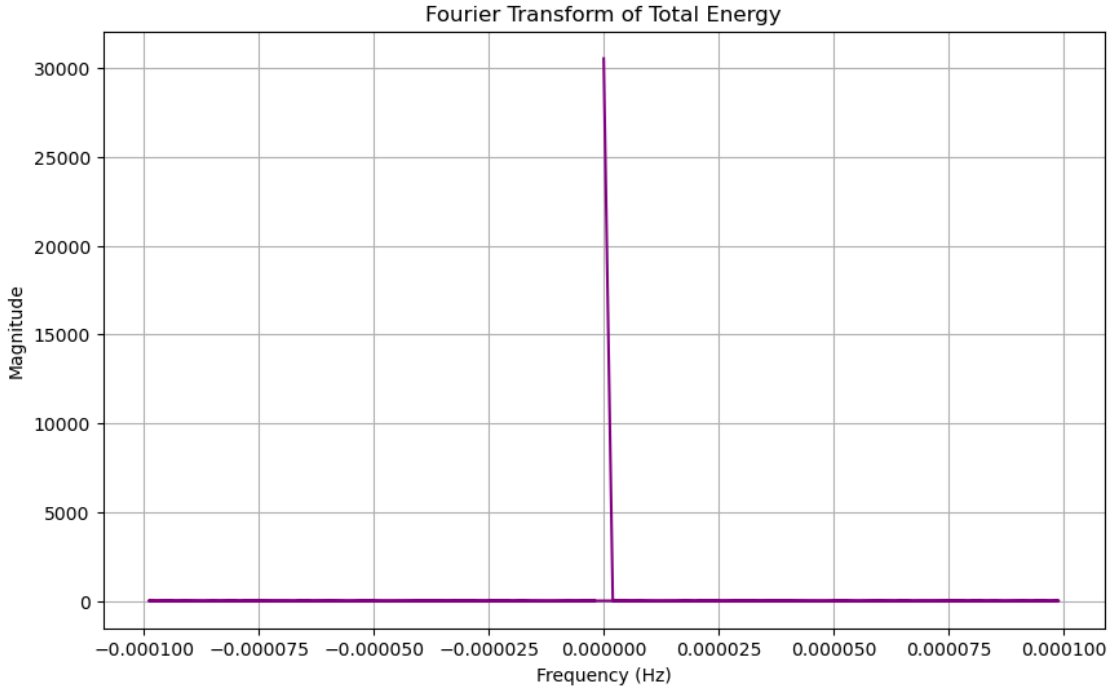
[5]:
```python
import numpy as np
import matplotlib.pyplot as plt

# Load RDF data
rdf_data = np.loadtxt('coord1.dat')

# Separate data into r and g(r)
r = rdf_data[:, 0]       # First column is r
g_r = rdf_data[:, 1]     # Second column is g(r)

# Plot g(r) vs r
plt.figure(figsize=(8, 6))
plt.plot(r, g_r, label='Radial Distribution Function')
plt.xlabel('Distance r (Å)')
plt.ylabel('g(r)')
plt.title('Radial Distribution Function g(r) vs. r')
plt.legend()
plt.grid(True)
plt.show()
```

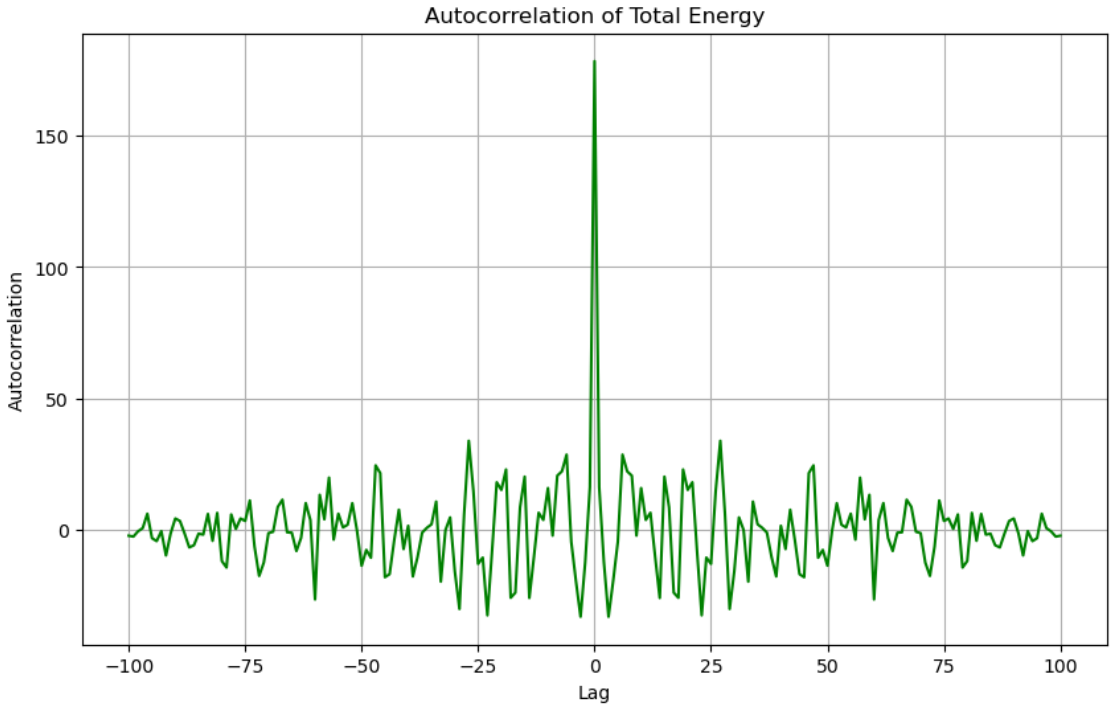


[10]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.fft import fft, fftfreq
from scipy.signal import correlate

# Load data function
def load_energy_data(filename):
    data = []
    with open(filename, 'r') as file:
        for line in file:
            if line.startswith("ENERGY:"):
                values = line.split()[1:]
                data.append([float(val) for val in values])
    return np.array(data)

# Load and extract relevant data
data = load_energy_data('prod_ener.txt')
time_step = data[:, 0]
total_energy = data[:, 11]  # Assuming the total energy is in column 11

# 1. Energy Trend Analysis
mean_energy = np.mean(total_energy)  # kcal/mol
variance_energy = np.var(total_energy)  # kcal/mol^2

# Moving Average (window size of 10 steps for example)
window_size = 10
moving_avg_energy = np.convolve(total_energy, np.ones(window_size)/window_size,␣
 ↪mode='valid')

plt.figure(figsize=(10, 6))
plt.plot(time_step, total_energy, label='Total Energy', color='blue', lw=1)
plt.plot(time_step[window_size-1:], moving_avg_energy, label='Moving Average␣
 ↪(10 steps)', color='orange', lw=2)
plt.xlabel('Time Step')
plt.ylabel('Total Energy (kcal/mol)')
plt.title('Energy Trend Analysis')
plt.grid(True)
plt.legend()
plt.show()

# 2. Frequency Analysis (Fourier Transform)
energy_fft = fft(total_energy)
frequencies = fftfreq(len(total_energy), d=(time_step[1] - time_step[0]))

plt.figure(figsize=(10, 6))
plt.plot(frequencies, np.abs(energy_fft), color='purple')
plt.xlabel('Frequency (Hz)')
```

```
plt.ylabel('Magnitude')
plt.title('Fourier Transform of Total Energy')
plt.grid(True)
plt.show()

# 3. Autocorrelation Analysis
autocorrelation = correlate(total_energy - mean_energy, total_energy -␣
 ↪mean_energy, mode='full')
lags = np.arange(-len(total_energy) + 1, len(total_energy))

plt.figure(figsize=(10, 6))
plt.plot(lags, autocorrelation, color='green')
plt.xlabel('Lag')
plt.ylabel('Autocorrelation')
plt.title('Autocorrelation of Total Energy')
plt.grid(True)
plt.show()

# 4. Bond Energy and Bond Strength Estimation (Simplified)
# Estimate fluctuations as a proxy for bond stability
fluctuations = total_energy - mean_energy

# Approximate bond energy (based on fluctuations, assuming kcal/mol units)
bond_energy_kcal_per_mol = np.mean(np.abs(fluctuations))  # kcal/mol

# Bond strength estimation (range of fluctuations in kcal/mol)
bond_strength_kcal_per_mol = np.max(np.abs(fluctuations)) - np.min(np.
 ↪abs(fluctuations))

# If you need to convert kcal/mol to kJ/mol
bond_energy_kJ_per_mol = bond_energy_kcal_per_mol * 4.184  # Conversion factor
bond_strength_kJ_per_mol = bond_strength_kcal_per_mol * 4.184  # Conversion␣
 ↪factor

# Print results in kcal/mol and kJ/mol
print(f"Mean Total Energy (kcal/mol): {mean_energy}")
print(f"Variance of Total Energy (kcal/mol^2): {variance_energy}")
print(f"Approximate Bond Energy (kcal/mol): {bond_energy_kcal_per_mol}")
print(f"Approximate Bond Strength (Range of Fluctuations in kcal/mol):␣
 ↪{bond_strength_kcal_per_mol}")

print(f"Approximate Bond Energy (kJ/mol): {bond_energy_kJ_per_mol}")
print(f"Approximate Bond Strength (Range of Fluctuations in kJ/mol):␣
 ↪{bond_strength_kJ_per_mol}")
```


Energy Trend Analysis
Total Energy
Moving Average (10 steps)
Total Energy (kcal/mol)
306
305
304
303
302
301
300
299
298
0
100000
200000
300000
400000
500000
Time Step


Fourier Transform of Total Energy
Magnitude
30000
25000
20000
15000
10000
5000
0
−0.000100
−0.000075
−0.000050
−0.000025
0.000000
0.000025
0.000050
0.000075
0.000100
Frequency (Hz)




```
Mean Total Energy (kcal/mol): 302.27101881188116
Variance of Total Energy (kcal/mol^2): 1.7672740203391823
Approximate Bond Energy (kcal/mol): 1.0520537790412687
Approximate Bond Strength (Range of Fluctuations in kcal/mol):
4.3308999999999855
Approximate Bond Energy (kJ/mol): 4.401793011508668
Approximate Bond Strength (Range of Fluctuations in kJ/mol): 18.12048559999994
```

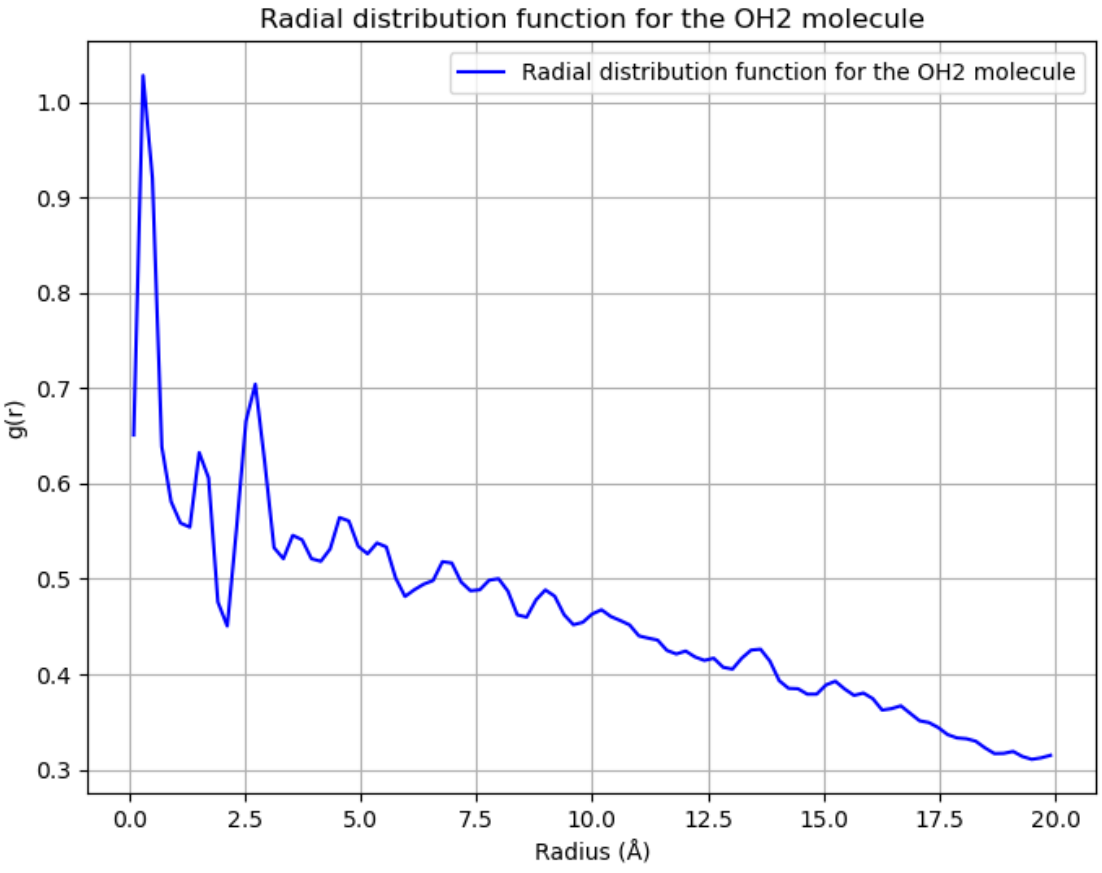
[11]:
```python
import numpy as np

# Given constants for O-H bond in water (can be adjusted based on your force␣
 ↪field)
k_bond = 450.0  # Bond force constant in kcal/mol/Å^2 (typical for O-H bond)
r_eq = 0.96     # Equilibrium bond length in Å (typical for O-H bond in water)

# Example function to calculate bond energy from harmonic potential
def calculate_bond_energy(r, k_bond, r_eq):
    """Calculate bond energy for a given bond length r using the harmonic␣
 ↪potential."""
    return 0.5 * k_bond * (r - r_eq)**2

# Assuming you have the bond lengths from your simulation (you would extract␣
 ↪these from the simulation trajectory)
# For demonstration, let's assume we have an array of O-H bond lengths (in Å):
```

```
# Let's say 'bond_lengths' is a NumPy array of bond lengths in Å for O-H bonds␣
 ↪over time
bond_lengths = np.array([0.95, 0.97, 0.96, 0.94, 0.98])  # Example values

# Calculate bond energies for each O-H bond length
bond_energies = calculate_bond_energy(bond_lengths, k_bond, r_eq)

# Now we can average bond energy over all frames (for a simulation)
mean_bond_energy = np.mean(bond_energies)

# Output the result
print(f"Average O-H Bond Energy (kcal/mol): {mean_bond_energy}")
```

```
Average O-H Bond Energy (kcal/mol): 0.04500000000000008
```

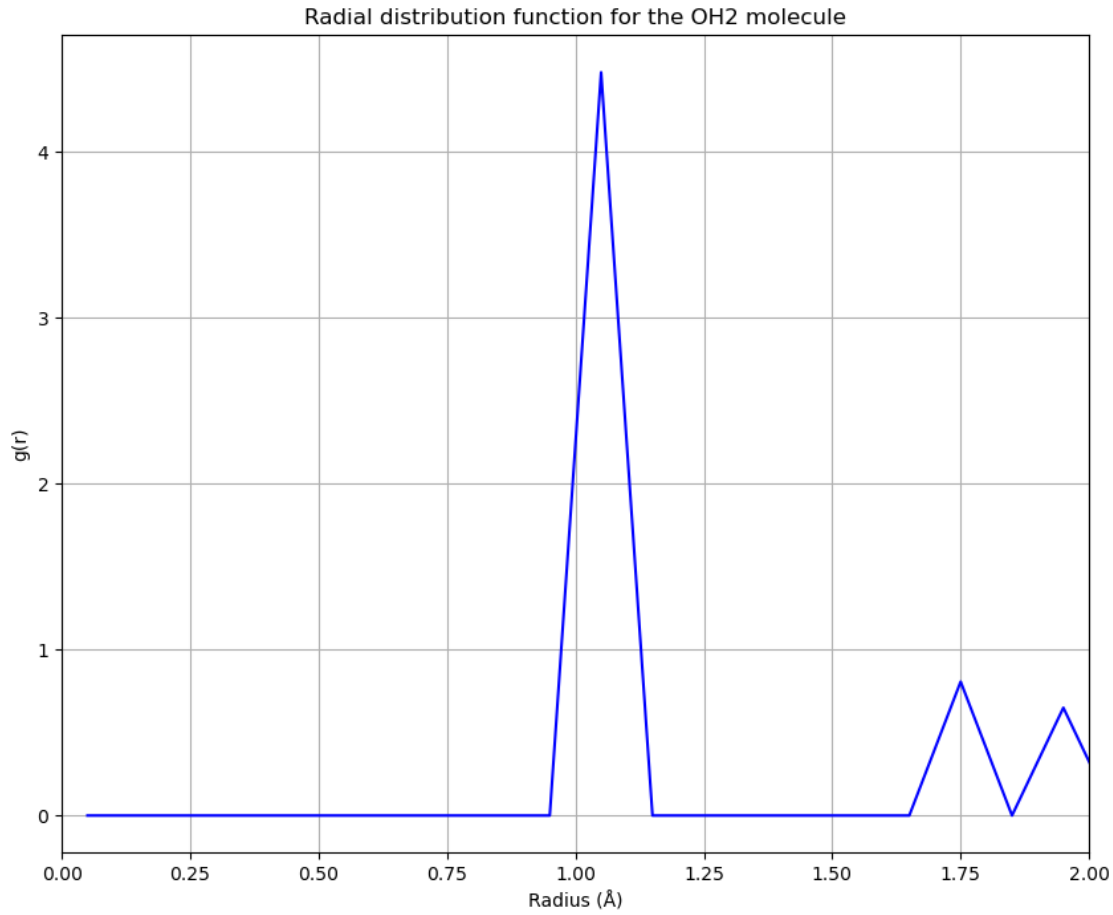
[103]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.spatial.distance import pdist
from scipy.ndimage import gaussian_filter1d  # for smoothing

# Load the data
data = np.loadtxt('coord1.dat')
x_coords = data[:, 2]
y_coords = data[:, 3]
z_coords = data[:, 4]

# Number of atoms and density estimation
num_atoms = len(x_coords)
volume = (np.ptp(x_coords) * np.ptp(y_coords) * np.ptp(z_coords))  #␣
 ↪approximate box volume
density = num_atoms / volume  # density

# Calculate pairwise distances
pairwise_distances = pdist(np.column_stack((x_coords, y_coords, z_coords)))

# Define bin edges and width
bin_edges = np.linspace(0, 20, 100)  # Adjust range and bin count as needed
dr = bin_edges[1] - bin_edges[0]

# Compute histogram (count of pairs within each spherical shell)
hist, bin_edges = np.histogram(pairwise_distances, bins=bin_edges)
r_values = 0.5 * (bin_edges[1:] + bin_edges[:-1])

# Volume of each spherical shell
shell_volumes = 4 * np.pi * (r_values**2) * dr

# Calculate g(r)
```

```
g_r = hist / (shell_volumes * density * num_atoms)

# Apply smoothing for clarity
g_r_smooth = gaussian_filter1d(g_r, sigma=1)

# Plot g(r) vs. r
plt.figure(figsize=(8, 6))
plt.plot(r_values, g_r_smooth, label="Radial distribution function for the OH2␣
 ↪molecule", color='b')
plt.xlabel("Radius (Å)")
plt.ylabel("g(r)")
plt.title("Radial distribution function for the OH2 molecule")
plt.grid(True)
plt.legend()
plt.show()
```



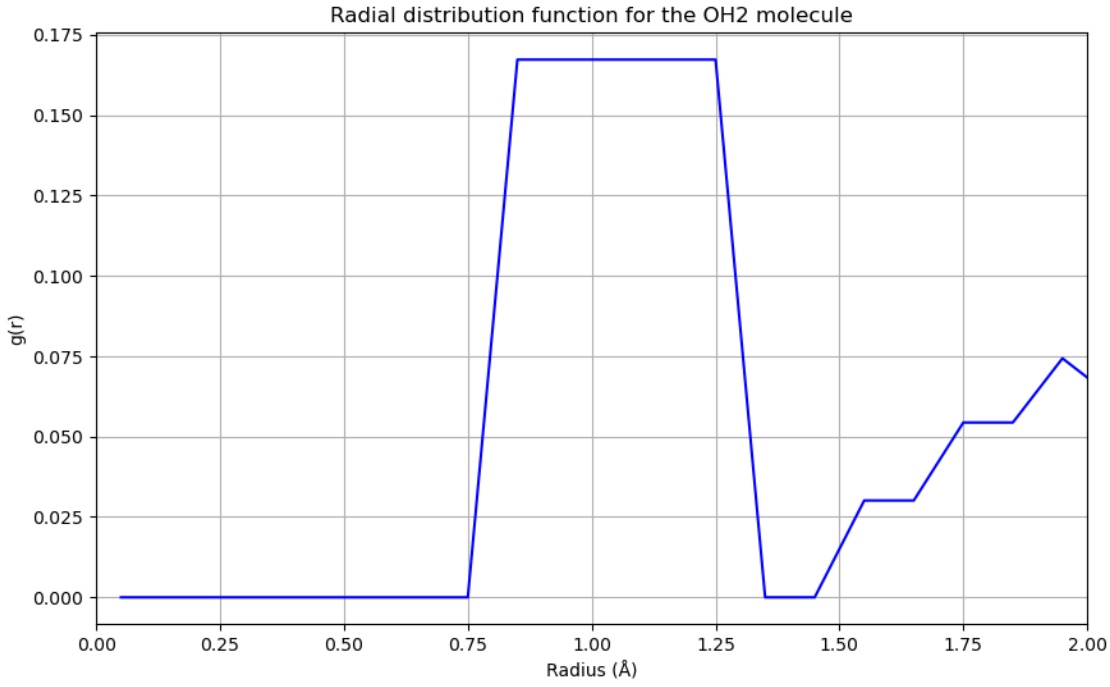
[23]:
```
import numpy as np
import matplotlib.pyplot as plt
```

```
# Load your data (time frame, atom ID, x, y, z) - adjust the file path as needed
data = np.loadtxt('coord1.dat')
oxygen_positions = data[data[:, 1] % 3 == 1, 2:5]  # Select oxygen positions␣
 ↪assuming IDs modulo 3 are oxygen

# Parameters for g(r) calculation
r_max = 35.0  # Maximum radius to consider
bin_width = 0.1  # Width of each radial shell
bins = np.arange(0, r_max, bin_width)  # Bin edges
g_r = np.zeros(len(bins) - 1)

# Density estimation (number of particles per unit volume)
volume = (4/3) * np.pi * (r_max**3)
density = len(oxygen_positions) / volume

# Calculate g(r)
for i in range(len(oxygen_positions)):
    for j in range(i + 1, len(oxygen_positions)):
        dist = np.linalg.norm(oxygen_positions[i] - oxygen_positions[j])
        if dist < r_max:
            bin_index = int(dist / bin_width)
            if bin_index < len(g_r):  # Check to avoid out-of-bounds error
                g_r[bin_index] += 2  # Count each pair once


# Normalize g(r)
shell_volumes = (4/3) * np.pi * (np.diff(bins**3))
g_r /= (shell_volumes * density * len(oxygen_positions))
g_r
# Plot the radial distribution function
plt.figure(figsize=(10, 8))
plt.xlim(0,2)
plt.plot(bins[:-1] + bin_width / 2, g_r, color='b')
plt.xlabel('Radius (Å)')
plt.ylabel('g(r)')
plt.title('Radial distribution function for the OH2 molecule')
plt.grid(True)
plt.show()
```



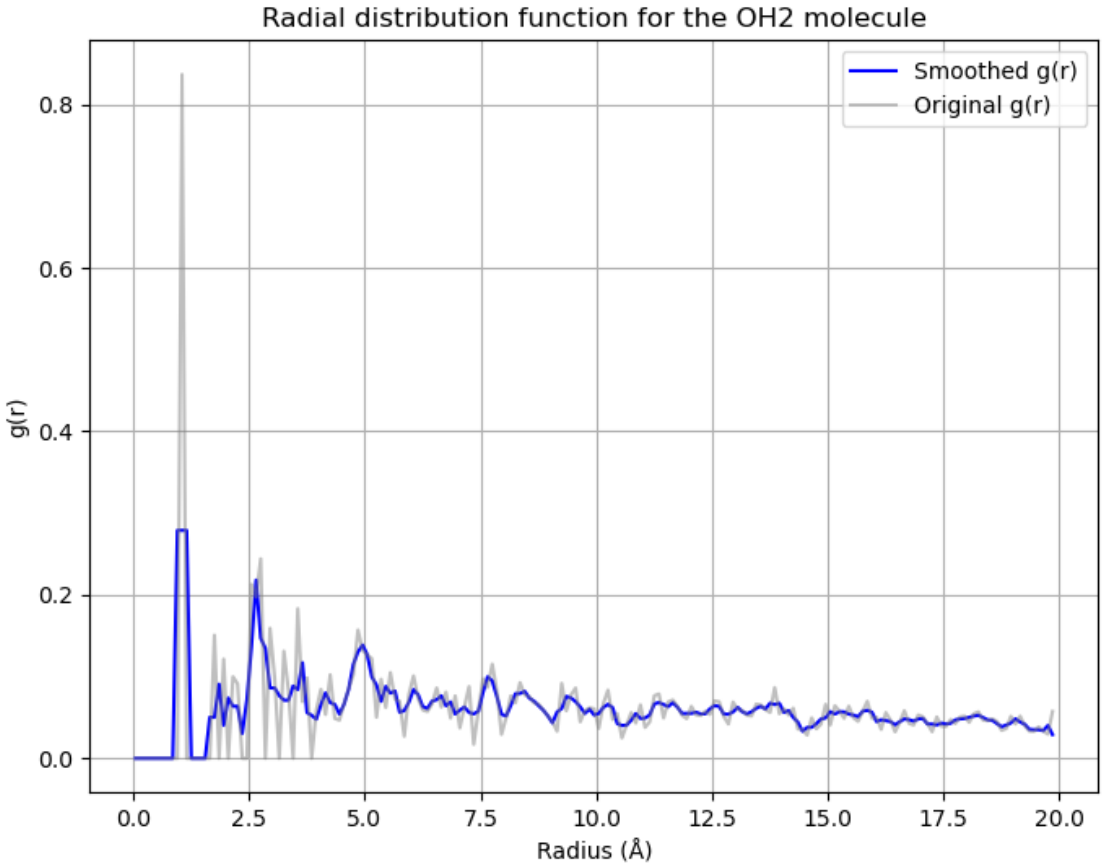
[19]:
```
import numpy as np
import matplotlib.pyplot as plt

# Load your data (time frame, atom ID, x, y, z) - adjust the file path as needed
data = np.loadtxt('coord1.dat')
oxygen_positions = data[data[:, 1] % 3 == 1, 2:5]  # Select oxygen positions␣
 ↪assuming IDs modulo 3 are oxygen

# Parameters for g(r) calculation
r_max = 20.0  # Maximum radius to consider
bin_width = 0.1  # Width of each radial shell
bins = np.arange(0, r_max, bin_width)  # Bin edges
g_r = np.zeros(len(bins) - 1)

# Density estimation (number of particles per unit volume)
volume = (4/3) * np.pi * (r_max**3)
density = len(oxygen_positions) / volume
```

```python
# Calculate g(r)
for i in range(len(oxygen_positions)):
    for j in range(i + 1, len(oxygen_positions)):
        dist = np.linalg.norm(oxygen_positions[i] - oxygen_positions[j])
        if dist < r_max:
            bin_index = int(dist / bin_width)
            if bin_index < len(g_r):  # Check to avoid out-of-bounds error
                g_r[bin_index] += 2  # Count each pair once

# Normalize g(r)
shell_volumes = (4/3) * np.pi * (np.diff(bins**3))
g_r /= (shell_volumes * density * len(oxygen_positions))

# Apply Moving Average for smoothing
window_size = 5  # Set the size of the moving average window
g_r_smooth = np.convolve(g_r, np.ones(window_size)/window_size, mode='same')

# Plot the smoothed radial distribution function
plt.figure(figsize=(10, 6))
plt.plot(bins[:-1] + bin_width / 2, g_r_smooth, color='b')
plt.xlim(0,2)
plt.xlabel('Radius (Å)')
plt.ylabel('g(r)')
plt.title('Radial distribution function for the OH2 molecule')
plt.grid(True)
plt.show()
```



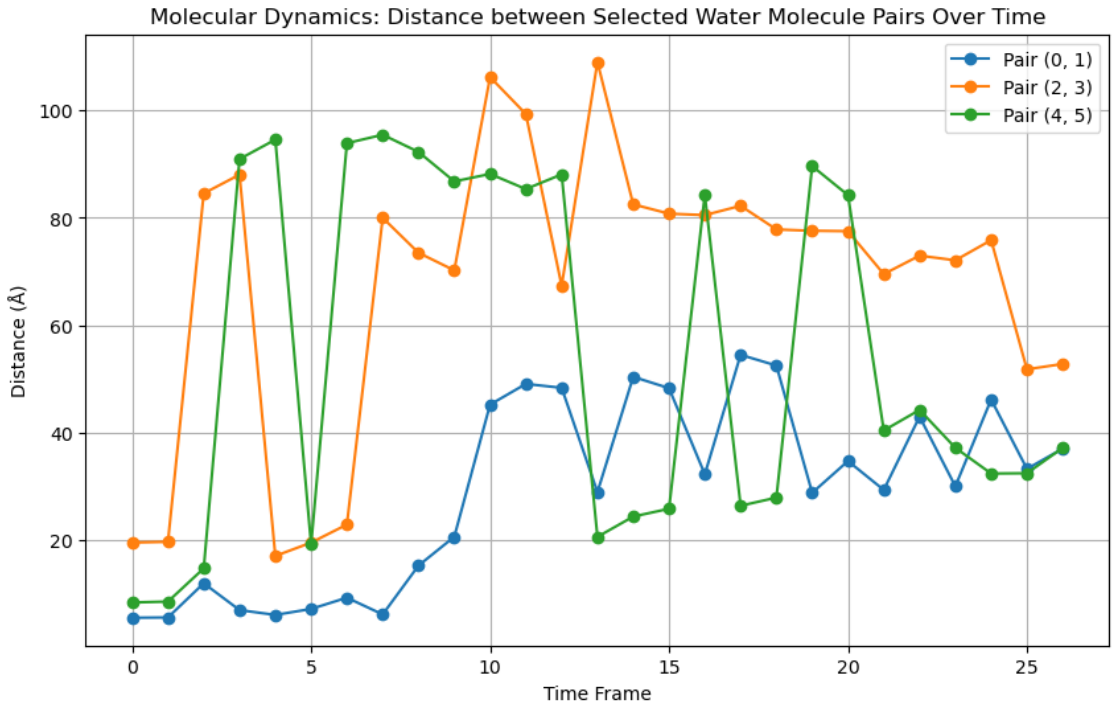
[22]:
```
import numpy as np
import matplotlib.pyplot as plt

# Load your data (time frame, atom ID, x, y, z) - adjust the file path as needed
data = np.loadtxt('coord1.dat')
oxygen_positions = data[data[:, 1] % 3 == 1, 2:5]  # Select oxygen positions␣
 ↪assuming IDs modulo 3 are oxygen

# Parameters for g(r) calculation
r_max = 20.0  # Maximum radius to consider
bin_width = 0.1  # Width of each radial shell
bins = np.arange(0, r_max, bin_width)  # Bin edges
g_r = np.zeros(len(bins) - 1)

# Calculate the density using only the volume within r_max
num_oxygen = len(oxygen_positions)
volume = (4/3) * np.pi * (r_max**3)
density = num_oxygen / volume

# Calculate g(r) by counting neighbors in spherical shells
for i in range(num_oxygen):
    for j in range(i + 1, num_oxygen):
        dist = np.linalg.norm(oxygen_positions[i] - oxygen_positions[j])
        if dist < r_max:
            bin_index = int(dist / bin_width)
            if bin_index < len(g_r):  # Check to avoid out-of-bounds error
                g_r[bin_index] += 2  # Count each pair once

# Normalize g(r) by shell volumes and density
shell_volumes = (4/3) * np.pi * ((bins[1:]**3) - (bins[:-1]**3))  # Volume of␣
 ↪each shell
g_r /= (shell_volumes * density * num_oxygen)

# Optional: Apply a very light smoothing if necessary
window_size = 3  # Small window to avoid excessive smoothing
g_r_smooth = np.convolve(g_r, np.ones(window_size) / window_size, mode='same')

# Plot the radial distribution function
plt.figure(figsize=(8, 6))
plt.plot(bins[:-1] + bin_width / 2, g_r_smooth, color='b', label='Smoothed␣
 ↪g(r)')
plt.plot(bins[:-1] + bin_width / 2, g_r, color='gray', alpha=0.5,␣
 ↪label='Original g(r)')
```

```
plt.xlabel('Radius (Å)')
plt.ylabel('g(r)')
plt.title('Radial distribution function for the OH2 molecule')
plt.legend()
plt.grid(True)
plt.show()
```



[24]:
```
import numpy as np
import matplotlib.pyplot as plt

# Load the data
# Assuming data format: [time_frame, atom_ID, x, y, z]
data = np.loadtxt('coord.dat')

# Get unique time frames in the data
time_frames = np.unique(data[:, 0])

# Select oxygen atom positions for each time frame
```

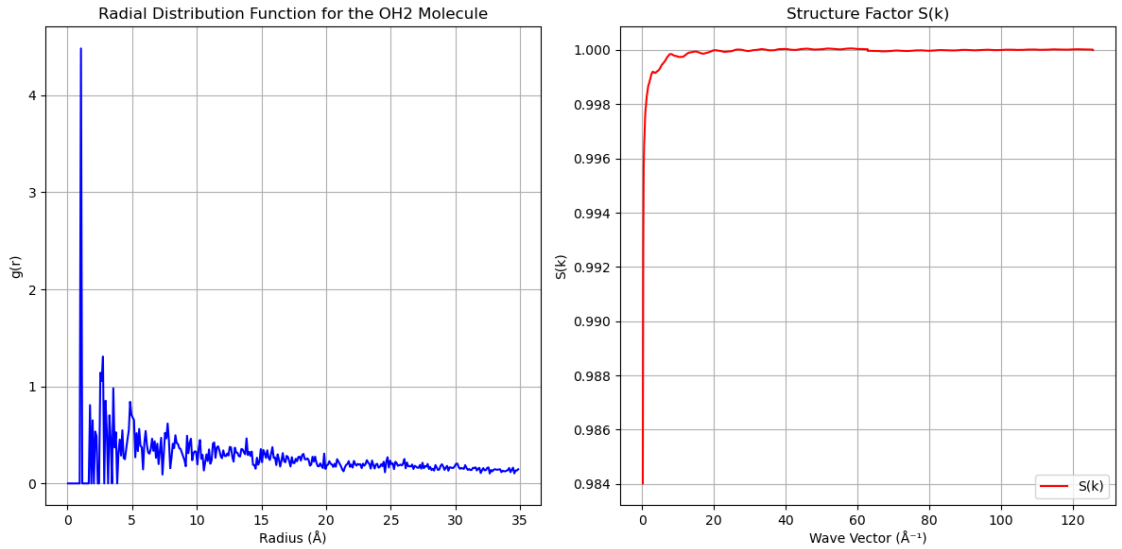
```
# Assuming atom IDs are structured so oxygen IDs are identifiable by modulo␣
 ↪operation (e.g., ID % 3 == 1)
oxygen_data = data[data[:, 1] % 3 == 1]

# Group oxygen positions by time frames
oxygen_positions_by_time = {t: oxygen_data[oxygen_data[:, 0] == t][:, 2:5] for␣
 ↪t in time_frames}

# Select 3 random pairs of oxygen atoms for distance tracking
num_pairs = 3  # Number of pairs to track
pairs = [(0, 1), (2, 3), (4, 5)]  # Example pairs; adjust according to your␣
 ↪data's atom ordering

# Initialize a dictionary to hold distance data for each pair
distances_over_time = {pair: [] for pair in pairs}

# Calculate distances between selected pairs at each time frame
for t in time_frames:
    positions = oxygen_positions_by_time[t]
    for pair in pairs:
        i, j = pair
        dist = np.linalg.norm(positions[i] - positions[j])
        distances_over_time[pair].append(dist)

# Plotting the distances over time for each selected pair
plt.figure(figsize=(10, 6))

for idx, (pair, distances) in enumerate(distances_over_time.items()):
    plt.plot(time_frames, distances, label=f'Pair {pair}', marker='o',␣
 ↪linestyle='-')

plt.xlabel('Time Frame')
plt.ylabel('Distance (Å)')
plt.title('Molecular Dynamics: Distance between Selected Water Molecule Pairs␣
 ↪Over Time')
plt.legend()
plt.grid(True)
plt.show()
```



[44]:
```
import numpy as np
import matplotlib.pyplot as plt

# Load your data (time frame, atom ID, x, y, z) - adjust the file path as needed
data = np.loadtxt('coord1.dat')
oxygen_positions = data[data[:, 1] % 3 == 1, 2:5]  # Select oxygen positions␣
 ↪assuming IDs modulo 3 are oxygen

# Parameters for g(r) calculation
r_max = 35.0  # Maximum radius to consider
bin_width = 0.1  # Width of each radial shell
bins = np.arange(0, r_max, bin_width)  # Bin edges
g_r = np.zeros(len(bins) - 1)

# Calculate the density using only the volume within r_max
num_oxygen = len(oxygen_positions)
volume = (4/3) * np.pi * (r_max**3)
density = num_oxygen / volume

# Calculate g(r) by counting neighbors in spherical shells
for i in range(num_oxygen):
    for j in range(i + 1, num_oxygen):
        dist = np.linalg.norm(oxygen_positions[i] - oxygen_positions[j])
        if dist < r_max:
```

```
            bin_index = int(dist / bin_width)
            if bin_index < len(g_r):  # Check to avoid out-of-bounds error
                g_r[bin_index] += 2  # Count each pair once

# Normalize g(r) by shell volumes and density
shell_volumes = (4/3) * np.pi * ((bins[1:]**3) - (bins[:-1]**3))  # Volume of␣
 ↪each shell
g_r /= (shell_volumes * density * num_oxygen)

# Calculate the structure factor S(k)
k_max = 2 * np.pi / (bin_width / 2)  # Max k based on the bin width
k_values = np.linspace(0, k_max, 1000)  # Define k range
S_k = np.zeros_like(k_values)

# Calculate S(k) using numerical integration
for i, k in enumerate(k_values):
    integrand = (g_r - 1) * np.sin(k * bins[:-1]) / (k * bins[:-1])
    integrand[0] = 0  # Avoid division by zero at the origin
    # Perform numerical integration using the trapezoidal rule
    S_k[i] = 1 + density * np.trapz(integrand, bins[:-1])

# Plot the radial distribution function
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(bins[:-1] + bin_width / 2, g_r, color='b', label='g(r)')
plt.xlabel('Radius (Å)')
plt.ylabel('g(r)')
plt.title('Radial Distribution Function for the OH2 Molecule')
plt.grid(True)

# Plot the structure factor S(k)
plt.subplot(1, 2, 2)
plt.plot(k_values, S_k, color='r', label='S(k)')
plt.xlabel('Wave Vector (Å ⁻¹)')
plt.ylabel('S(k)')
plt.title('Structure Factor S(k)')
plt.grid(True)
plt.legend()

plt.tight_layout()
plt.show()
```

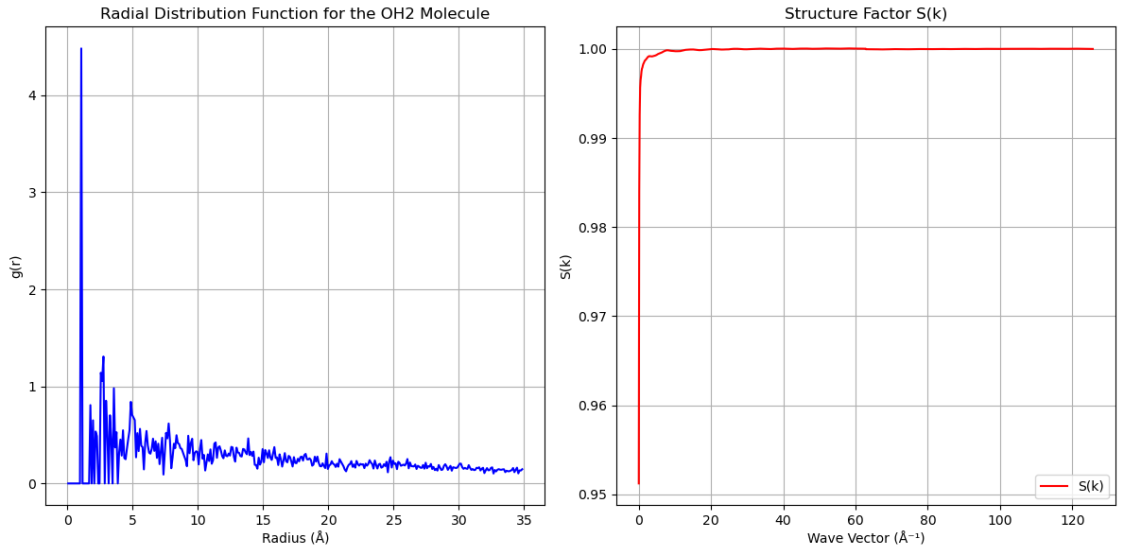
```
C:\Users\Katha\AppData\Local\Temp\ipykernel_7824\3412805694.py:39:
RuntimeWarning: invalid value encountered in divide
  integrand = (g_r - 1) * np.sin(k * bins[:-1]) / (k * bins[:-1])
```



[55]:
```
import numpy as np
import matplotlib.pyplot as plt

# Load your data (time frame, atom ID, x, y, z) - adjust the file path as needed
data = np.loadtxt('coord1.dat')
oxygen_positions = data[data[:, 1] % 3 == 1, 2:5]  # Select oxygen positions␣
 ↪assuming IDs modulo 3 are oxygen

# Parameters for g(r) calculation
r_max = 35.0  # Maximum radius to consider
bin_width = 0.1  # Width of each radial shell
bins = np.arange(0, r_max, bin_width)  # Bin edges
g_r = np.zeros(len(bins) - 1)

# Calculate the density using only the volume within r_max
num_oxygen = len(oxygen_positions)
volume = (4/3) * np.pi * (r_max**3)
density = num_oxygen / volume

# Calculate g(r) by counting neighbors in spherical shells
for i in range(num_oxygen):
    for j in range(i + 1, num_oxygen):
        dist = np.linalg.norm(oxygen_positions[i] - oxygen_positions[j])
        if dist < r_max:
            bin_index = int(dist / bin_width)
            if bin_index < len(g_r):  # Check to avoid out-of-bounds error
                g_r[bin_index] += 2  # Count each pair once
```

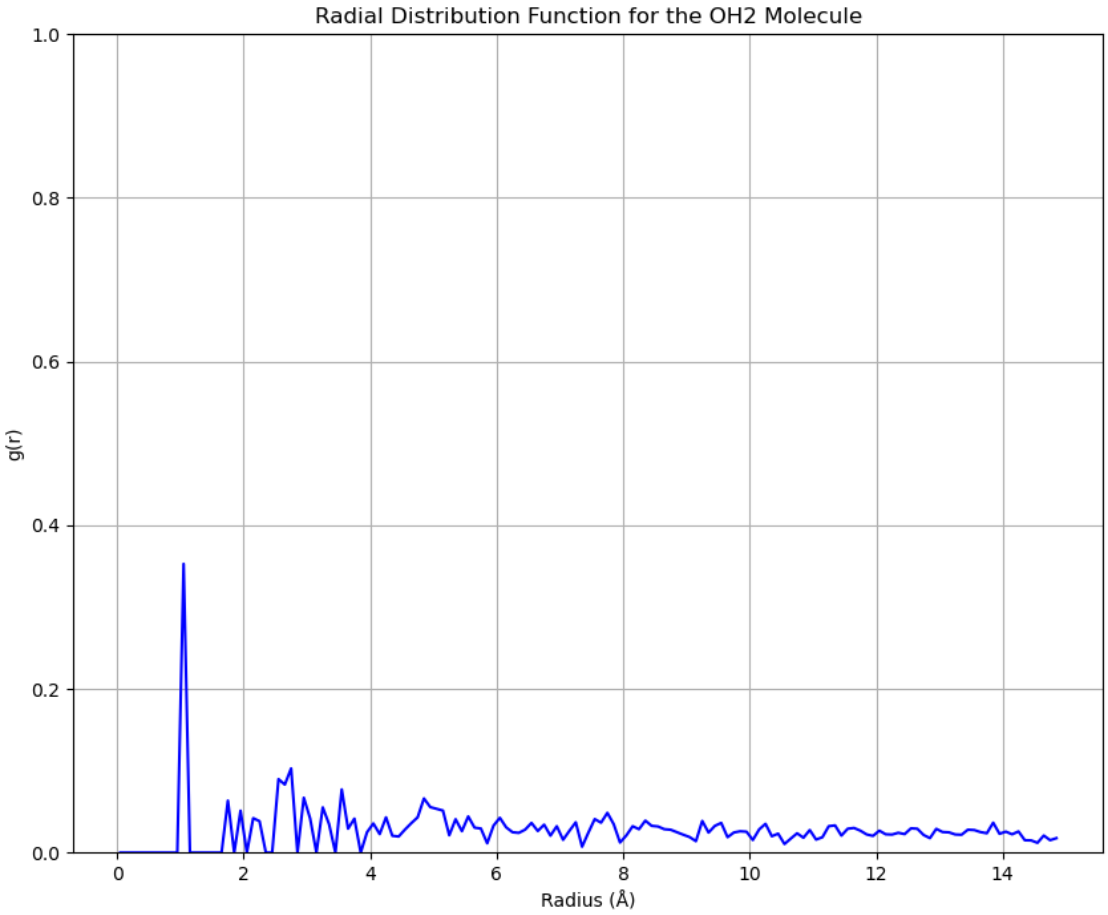
```
# Normalize g(r) by shell volumes and density
shell_volumes = (4/3) * np.pi * ((bins[1:]**3) - (bins[:-1]**3))  # Volume of␣
 ↪each shell
g_r /= (shell_volumes * density * num_oxygen)

# Calculate the structure factor S(k)
k_max = 2 * np.pi / (bin_width / 2)  # Max k based on the bin width
k_values = np.linspace(0, k_max, 1000)  # Define k range
S_k = np.zeros_like(k_values)

# Calculate S(k) using numerical integration
for i, k in enumerate(k_values):
    if k == 0:
        S_k[i] = 1 + density * np.sum(g_r - 1) * bin_width  # Handle k = 0 case␣
 ↪directly
    else:
        integrand = (g_r - 1) * np.sin(k * bins[:-1]) / (k * bins[:-1])
        integrand[0] = 0  # Avoid division by zero at the origin
        # Perform numerical integration using the trapezoidal rule
        S_k[i] = 1 + density * np.trapz(integrand, bins[:-1])

# Plot the radial distribution function
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(bins[:-1] + bin_width / 2, g_r, color='b', label='g(r)')
plt.xlabel('Radius (Å)')
plt.ylabel('g(r)')
plt.title('Radial Distribution Function for the OH2 Molecule')
plt.grid(True)

# Plot the structure factor S(k)
plt.subplot(1, 2, 2)
plt.plot(k_values, S_k, color='r', label='S(k)')
plt.xlabel('Wave Vector (Å ⁻¹)')
plt.ylabel('S(k)')
plt.title('Structure Factor S(k)')
plt.grid(True)
plt.legend()

plt.tight_layout()
plt.show()
```

```
C:\Users\Katha\AppData\Local\Temp\ipykernel_7824\811117546.py:42:
RuntimeWarning: invalid value encountered in divide
  integrand = (g_r - 1) * np.sin(k * bins[:-1]) / (k * bins[:-1])
```



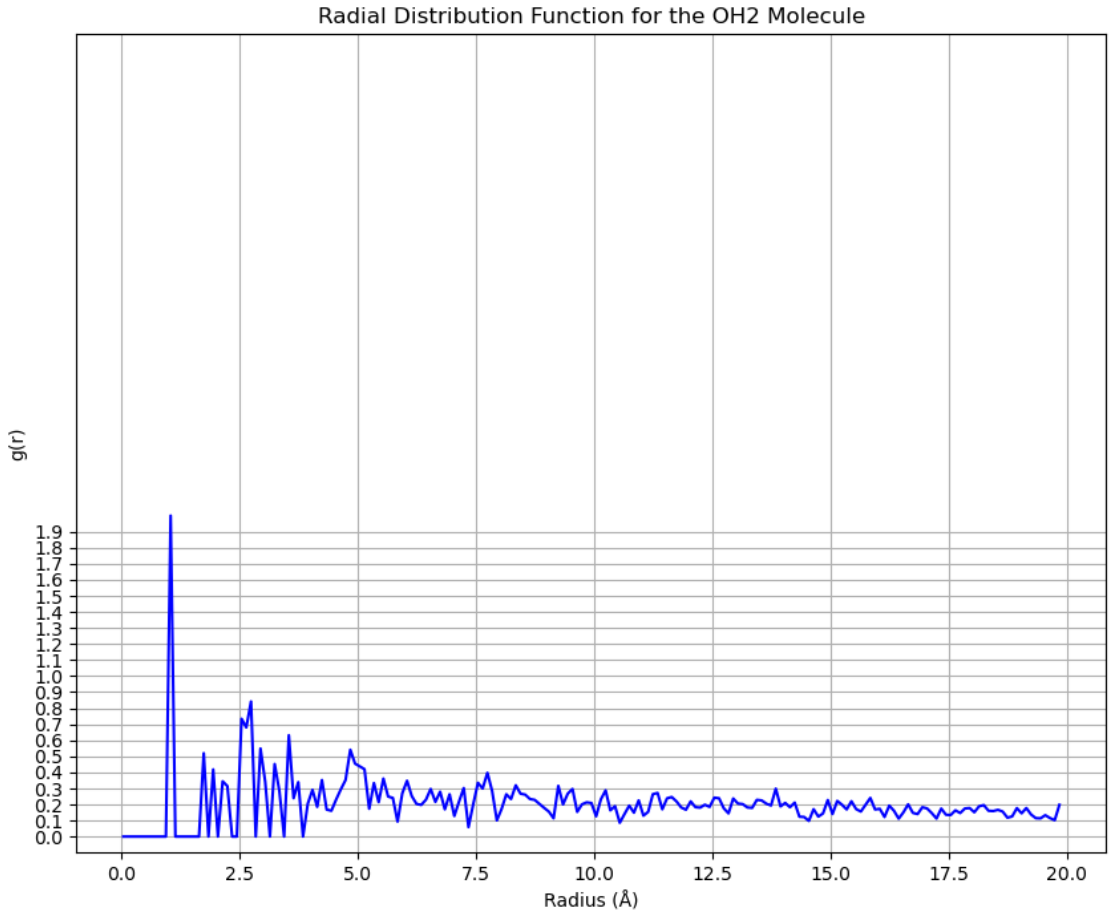
[2]:
```
import numpy as np
import matplotlib.pyplot as plt

# Load your data (time frame, atom ID, x, y, z) - adjust the file path as needed
data = np.loadtxt('coord1.dat')
oxygen_positions = data[data[:, 1] % 3 == 1, 2:5]  # Select oxygen positions␣
 ↪assuming IDs modulo 3 are oxygen

# Parameters for g(r) calculation
r_max = 15.0  # Maximum radius to consider
bin_width = 0.1  # Width of each radial shell
bins = np.arange(0, r_max, bin_width)  # Bin edges
g_r = np.zeros(len(bins) - 1)

# Density estimation (number of particles per unit volume)
volume = (4/3) * np.pi * (r_max**3)
density = len(oxygen_positions) / volume

# Calculate g(r)
for i in range(len(oxygen_positions)):
    for j in range(i + 1, len(oxygen_positions)):
        dist = np.linalg.norm(oxygen_positions[i] - oxygen_positions[j])
        if dist < r_max:
            bin_index = int(dist / bin_width)
            if bin_index < len(g_r):  # Check to avoid out-of-bounds error
                g_r[bin_index] += 2  # Count each pair once

# Normalize g(r)
```

```
shell_volumes = (4/3) * np.pi * (np.diff(bins**3))
g_r /= (shell_volumes * density * len(oxygen_positions))

# Rescale g(r) to a specific range, if needed
# For example, let's say you want to limit it to 0 to 0.18
max_g_r = np.min([np.max(g_r), 2])  # Ensures g(r) does not exceed 0.18
g_r_scaled = np.clip(g_r, 0, max_g_r)

# Plot the radial distribution function
plt.figure(figsize=(10, 8))
plt.plot(bins[:-1] + bin_width / 2, g_r_scaled, color='b')  # Centered bin␣
 ↪edges for plotting
plt.xlabel('Radius (Å)')
plt.ylabel('g(r)')
plt.title('Radial Distribution Function for the OH2 Molecule')
plt.grid(True)

# Customize y-axis to show specific values
#y_ticks = np.arange(0, 0.2, 0.02)  # Ticks from 0 to 0.18 in steps of 0.02
#plt.yticks(y_ticks)

# Optionally set y-limits to improve visibility
plt.ylim(bottom=0, top=1)

plt.show()
```



[39]:
```
import numpy as np
import matplotlib.pyplot as plt

# Load your data (time frame, atom ID, x, y, z) - adjust the file path as needed
data = np.loadtxt('coord1.dat')
oxygen_positions = data[data[:, 1] % 3 == 1, 2:5]  # Select oxygen positions␣
 ↪assuming IDs modulo 3 are oxygen

# Parameters for g(r) calculation
r_max = 20.0  # Maximum radius to consider
bin_width = 0.1  # Width of each radial shell
bins = np.arange(0, r_max, bin_width)  # Bin edges
g_r = np.zeros(len(bins) - 1)

# Density estimation (number of particles per unit volume)
volume = (4/3) * np.pi * (r_max**3)
#density = len(oxygen_positions) / volume
```

```
# Calculate g(r)
for i in range(len(oxygen_positions)):
    for j in range(i + 1, len(oxygen_positions)):
        dist = np.linalg.norm(oxygen_positions[i] - oxygen_positions[j])
        if dist < r_max:
            bin_index = int(dist / bin_width)
            if bin_index < len(g_r):  # Check to avoid out-of-bounds error
                g_r[bin_index] += 2  # Count each pair once

# Normalize g(r)
shell_volumes = (4/3) * np.pi * (np.diff(bins**3))
g_r /= (shell_volumes)

# Rescale g(r) to a specific range, if needed
max_g_r = np.min([np.max(g_r), 2])  # Ensures g(r) does not exceed 2
g_r_scaled = np.clip(g_r, 0, max_g_r)

# Plot the radial distribution function
plt.figure(figsize=(10, 8))
plt.plot(bins[:-1] + bin_width / 2, g_r_scaled, color='b')  # Centered bin␣
 ↪edges for plotting
plt.xlabel('Radius (Å)')
plt.ylabel('g(r)')
plt.title('Radial Distribution Function for the OH2 Molecule')
plt.grid(True)

# Customize y-axis limits to scale up the lower part
plt.ylim(-0.1, 5)  # Adjust the upper limit if needed to allow for visibility

# Optionally, adjust the ticks for better clarity
plt.yticks(np.arange(0, 2, 0.1))  # Ticks from 0 to 1 in steps of 0.1

plt.show()
```

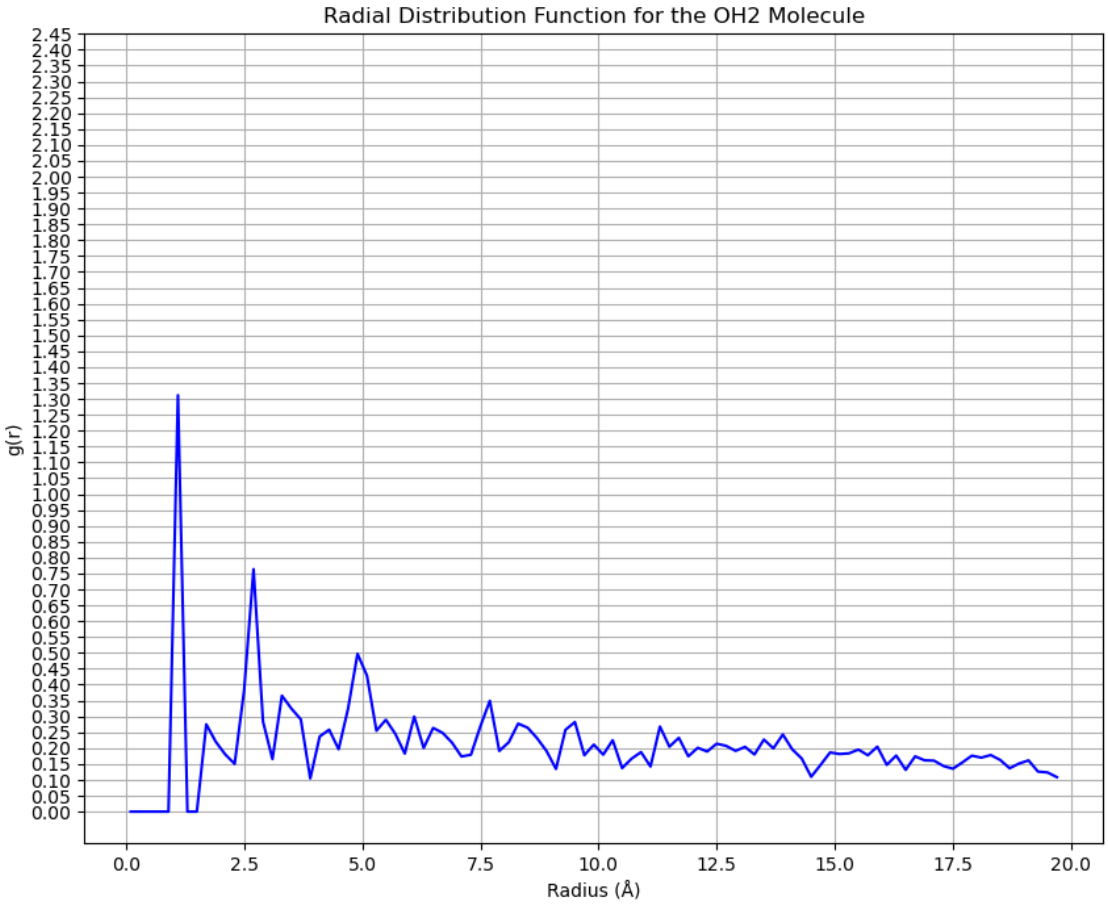


[3]:
```
import numpy as np
import matplotlib.pyplot as plt

# Load your data (time frame, atom ID, x, y, z) - adjust the file path as needed
data = np.loadtxt('coord1.dat')
oxygen_positions = data[data[:, 1] % 3 == 1, 2:5]  # Select oxygen positions␣
 ↪assuming IDs modulo 3 are oxygen

# Parameters for g(r) calculation
r_max = 20.0  # Maximum radius to consider
bin_width = 0.2  # Width of each radial shell
bins = np.arange(0, r_max, bin_width)  # Bin edges
g_r = np.zeros(len(bins) - 1)

# Density estimation (number of particles per unit volume)
volume = (4/3) * np.pi * (r_max**3)
#density = len(oxygen_positions) / volume
```

```
# Calculate g(r)
for i in range(len(oxygen_positions)):
    for j in range(i + 1, len(oxygen_positions)):
        dist = np.linalg.norm(oxygen_positions[i] - oxygen_positions[j])
        if dist < r_max:
            bin_index = int(dist / bin_width)
            if bin_index < len(g_r):  # Check to avoid out-of-bounds error
                g_r[bin_index] += 2  # Count each pair once

# Normalize g(r)
shell_volumes = (4/3) * np.pi * (np.diff(bins**3))
g_r /= (shell_volumes)
print((g_r))
# Rescale g(r) to a specific range, if needed
#max_g_r = np.min([np.max(g_r), 2])  # Ensures g(r) does not exceed 2
#g_r_scaled = np.clip(g_r, 0, max_g_r)

# Plot the radial distribution function
plt.figure(figsize=(10, 8))
plt.plot(bins[:-1] + bin_width / 2, g_r, color='b')  # Centered bin edges for␣
 ↪plotting
plt.xlabel('Radius (Å)')
plt.ylabel('g(r)')
plt.title('Radial Distribution Function for the OH2 Molecule')
plt.grid(True)

# Customize y-axis limits and ticks to focus on the range 0 to 1, with steps of␣
 ↪0.05
plt.ylim(-0.1, 2)  # Adjusted upper limit for better visibility
plt.yticks(np.arange(0,2.5, 0.05))  # Ticks from 0 to 0.2 in steps of 0.05

plt.show()
```

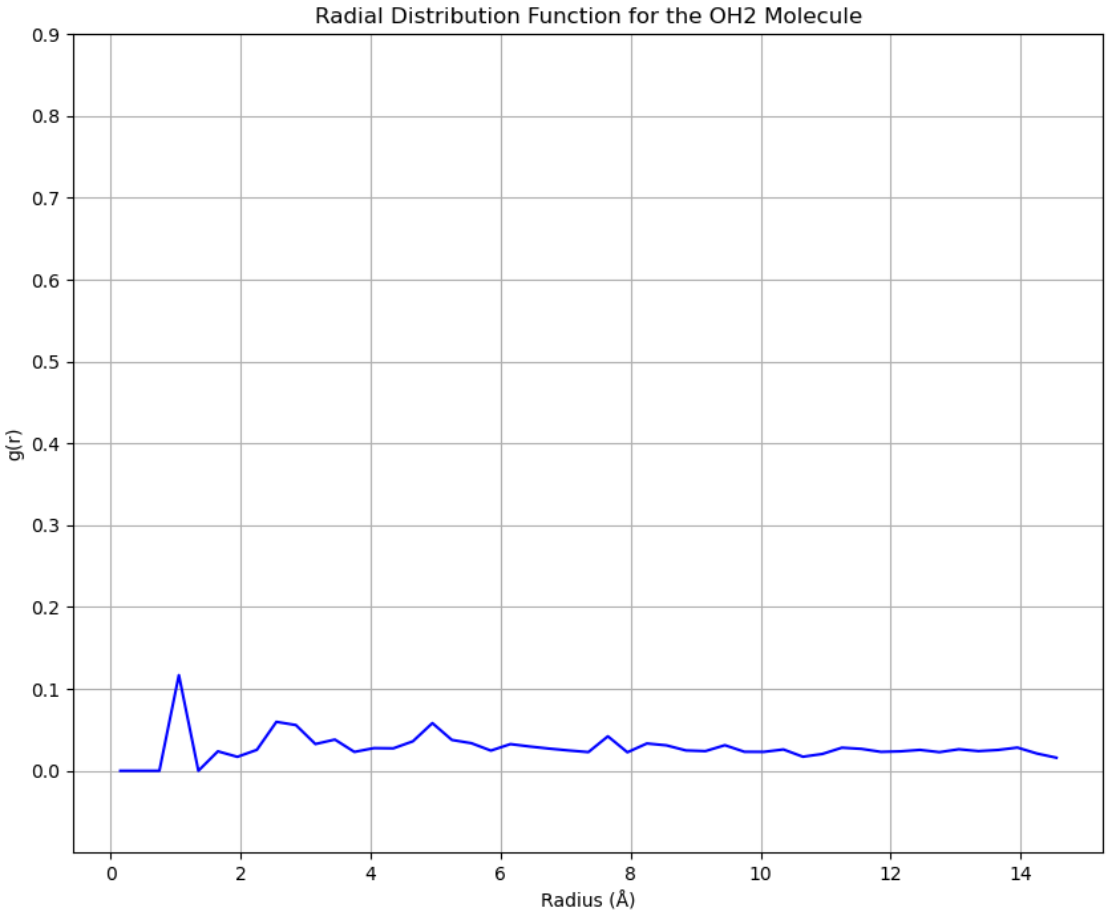
```
[0.         0.         0.         0.         0.         1.31171656
 0.         0.         0.27503734 0.22023285 0.18031149 0.15033527
 0.38176825 0.76376915 0.28375485 0.16555646 0.36525767 0.32471765
 0.29057012 0.10461543 0.2366499  0.25818214 0.19645525 0.32416894
 0.49708304 0.42827445 0.25493495 0.28934074 0.244904   0.18286665
 0.2993778  0.2004807  0.26366786 0.24816259 0.21727257 0.17363505
 0.1791837  0.26877909 0.34894551 0.19125113 0.21830845 0.27721976
 0.26432819 0.23128911 0.19087342 0.13452962 0.25761167 0.28214793
 0.17760311 0.21109518 0.17941594 0.22502113 0.13713617 0.16680965
 0.18753528 0.14208725 0.26797259 0.20457965 0.23252402 0.17419981
 0.20109961 0.18935329 0.21389968 0.20721587 0.19127667 0.20402895
 0.17994454 0.22704817 0.19926873 0.24299932 0.1961285  0.16733217
 0.10976029 0.14730204 0.18638649 0.18148187 0.1835671  0.19542167
 0.17756103 0.20459901 0.14735811 0.17671015 0.1315314  0.17405313
```

```
0.16159937 0.16056282 0.14357765 0.13511783 0.15494188 0.17633475
0.17003041 0.17821526 0.1627573  0.13653816 0.15148561 0.1614177
0.12604443 0.12347218 0.10867504]
```



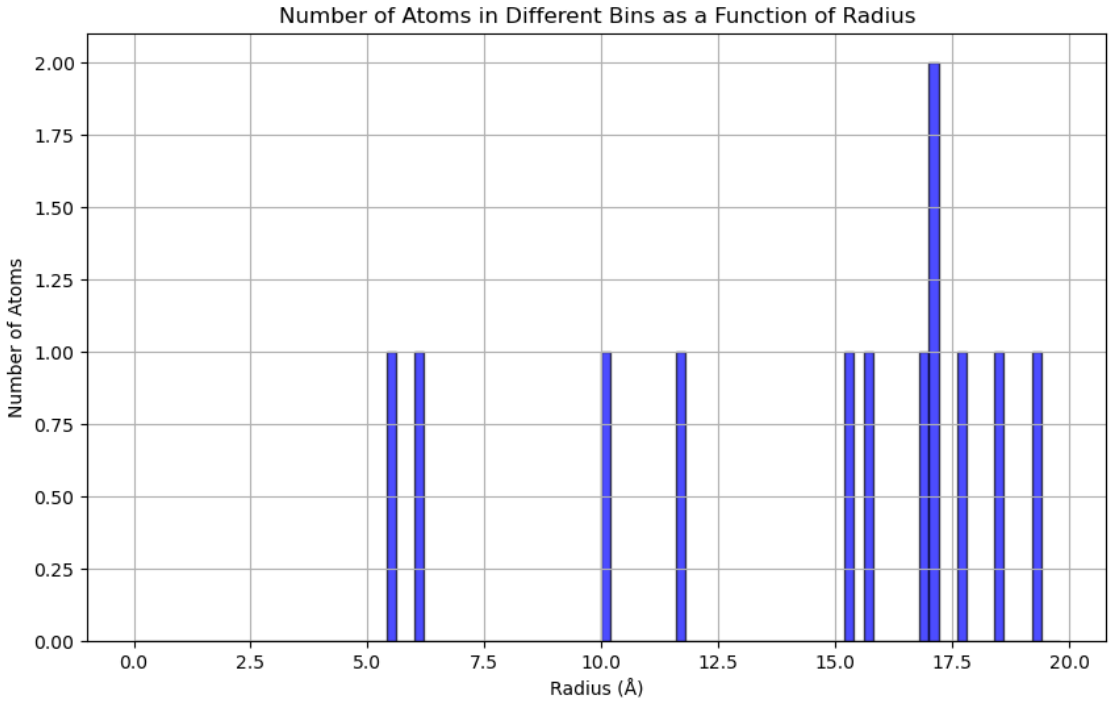
[28]:
```
import numpy as np
import matplotlib.pyplot as plt

# Load your data (time frame, atom ID, x, y, z) - adjust the file path as needed
data = np.loadtxt('coord1.dat')
oxygen_positions = data[data[:, 1] % 3 == 1, 2:5]  # Select oxygen positions␣
 ↪assuming IDs modulo 3 are oxygen

# Parameters for g(r) calculation
r_max = 15.0  # Maximum radius to consider
bin_width = 0.2  # Width of each radial shell
bins = np.arange(0, r_max, bin_width)  # Bin edges
g_r = np.zeros(len(bins) - 1)
```

```
# Density estimation (number of particles per unit volume)
volume = (4/3) * np.pi * (r_max**3)
density = len(oxygen_positions) / volume

# Calculate g(r)
for i in range(len(oxygen_positions)):
    for j in range(i + 1, len(oxygen_positions)):
        dist = np.linalg.norm(oxygen_positions[i] - oxygen_positions[j])
        if dist < r_max:
            bin_index = int(dist / bin_width)
            if bin_index < len(g_r):  # Check to avoid out-of-bounds error
                g_r[bin_index] += 2  # Count each pair once

# Normalize g(r)
shell_volumes = (4/3) * np.pi * (np.diff(bins**3))  # Volume of each shell
g_r /= (shell_volumes * density * len(oxygen_positions))  # Normalize by␣
 ↪density and number of oxygen atoms

# Plot the radial distribution function
plt.figure(figsize=(10, 8))
plt.plot(bins[:-1] + bin_width / 2, g_r, color='b')  # Centered bin edges for␣
 ↪plotting
plt.xlabel('Radius (Å)')
plt.ylabel('g(r)')
plt.title('Radial Distribution Function for the OH2 Molecule')
plt.grid(True)

# Customize y-axis limits and ticks
plt.ylim(-0.1, np.max(g_r) * 1.1)  # Set upper limit based on g(r) max value␣
 ↪for better visibility
plt.yticks(np.arange(0, np.ceil(np.max(g_r) * 1.1), 0.1))  # Adjust ticks based␣
 ↪on maximum g(r)

plt.show()
```



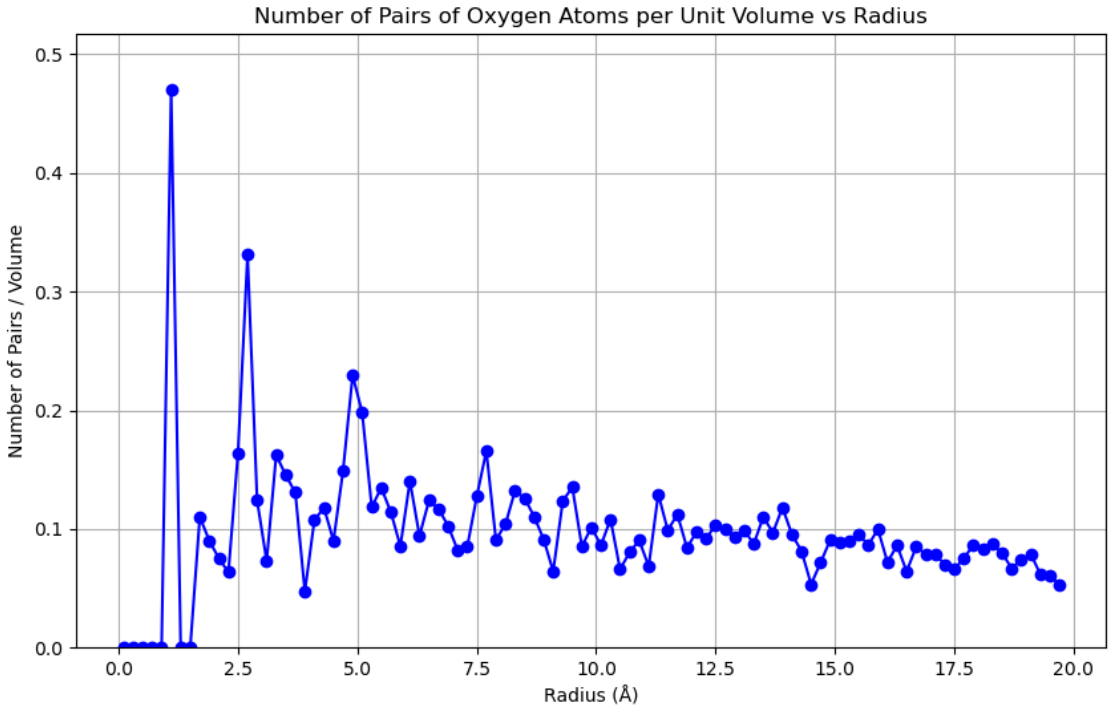
[29]:
```python
import numpy as np
import matplotlib.pyplot as plt

# Load your data (time frame, atom ID, x, y, z) - adjust the file path as needed
data = np.loadtxt('coord1.dat')
oxygen_positions = data[data[:, 1] % 3 == 1, 2:5]  # Select oxygen positions␣
 ↪assuming IDs modulo 3 are oxygen

# Parameters for the calculation
r_max = 20.0  # Maximum radius to consider
bin_width = 0.2  # Width of each radial shell
bins = np.arange(0, r_max, bin_width)  # Bin edges
atom_counts = np.zeros(len(bins) - 1)  # Initialize counts for each bin

# Count the number of atoms in each bin
for i in range(len(oxygen_positions)):
    for j in range(len(bins) - 1):
        # Calculate the distance from the atom to the reference origin (0,0,0)
```

```
        dist = np.linalg.norm(oxygen_positions[i])
        if bins[j] <= dist < bins[j + 1]:
            atom_counts[j] += 1  # Increment the count for the appropriate bin

# Calculate the bin centers for plotting
bin_centers = bins[:-1] + bin_width / 2

# Plot the number of atoms in different bins as a function of radius
plt.figure(figsize=(10, 6))
plt.bar(bin_centers, atom_counts, width=bin_width, color='b', alpha=0.7,␣
 ↪edgecolor='black')
plt.xlabel('Radius (Å)')
plt.ylabel('Number of Atoms')
plt.title('Number of Atoms in Different Bins as a Function of Radius')
plt.grid(True)
plt.show()
```



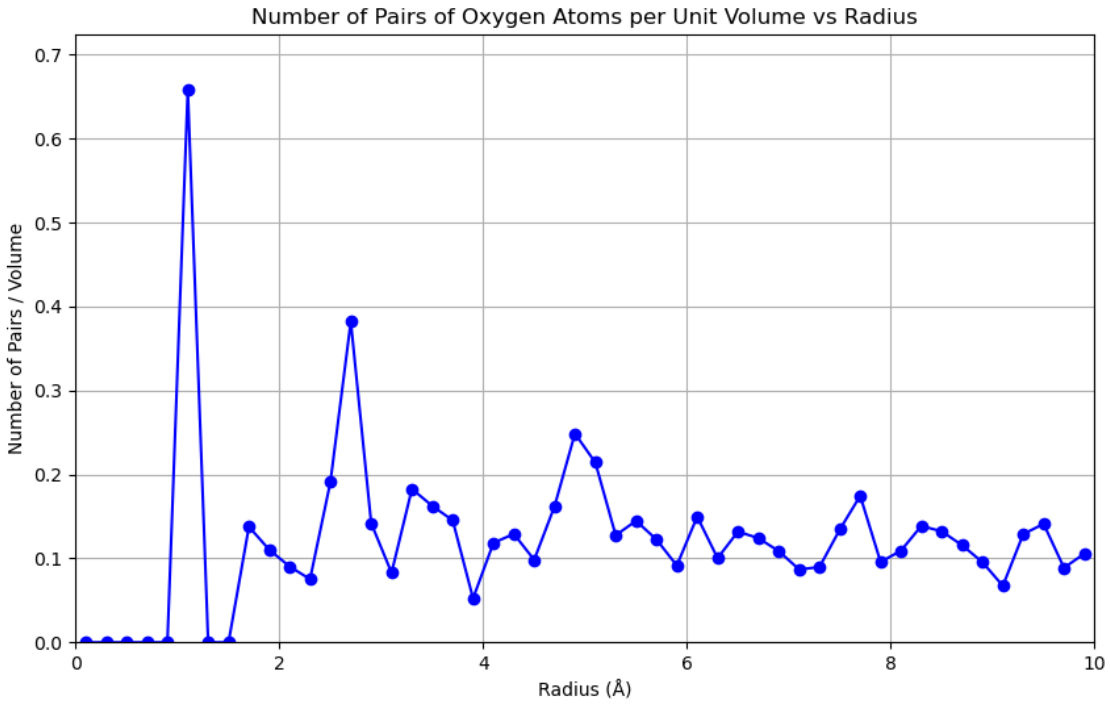
[31]:
```
import numpy as np
import matplotlib.pyplot as plt

# Load your data (time frame, atom ID, x, y, z) - adjust the file path as needed
data = np.loadtxt('coord1.dat')
oxygen_positions = data[data[:, 1] % 3 == 1, 2:5]  # Select oxygen positions␣
 ↪assuming IDs modulo 3 are oxygen
```

```
# Parameters for the calculation
r_max = 20.0  # Maximum radius to consider
bin_width = 0.2  # Width of each radial shell
bins = np.arange(0, r_max, bin_width)  # Bin edges
pair_counts = np.zeros(len(bins) - 1)  # Initialize counts for each bin

# Calculate pair counts in each bin
for i in range(len(oxygen_positions)):
    for j in range(i + 1, len(oxygen_positions)):
        dist = np.linalg.norm(oxygen_positions[i] - oxygen_positions[j])
        if dist < r_max:
            bin_index = int(dist / bin_width)
            if bin_index < len(pair_counts):  # Ensure we are within bounds
                pair_counts[bin_index] += 1  # Increment the count for the␣
 ↪appropriate bin

# Calculate the volume of each shell using the difference between spheres
shell_volumes = (4/3) * np.pi * ( (bins[1:] + bin_width) ** 3 - bins[1:] ** 3 )

# Calculate pairs per unit volume
pairs_per_volume = pair_counts / shell_volumes

# Calculate bin centers for plotting
bin_centers = bins[:-1] + bin_width / 2

# Plot number of pairs per unit volume vs radius
plt.figure(figsize=(10, 6))
plt.plot(bin_centers, pairs_per_volume, color='b', marker='o')
plt.xlabel('Radius (Å)')
plt.ylabel('Number of Pairs / Volume')
plt.title('Number of Pairs of Oxygen Atoms per Unit Volume vs Radius')
plt.grid(True)
plt.ylim(0, np.max(pairs_per_volume) * 1.1)  # Adjust y-axis for better␣
 ↪visibility
plt.show()
```



[32]:
```python
import numpy as np
import matplotlib.pyplot as plt

# Load your data (time frame, atom ID, x, y, z) - adjust the file path as needed
data = np.loadtxt('coord1.dat')
oxygen_positions = data[data[:, 1] % 3 == 1, 2:5]  # Select oxygen positions␣
 ↪assuming IDs modulo 3 are oxygen

# Parameters for the calculation
r_max = 10.0  # Maximum radius to consider
bin_width = 0.2  # Width of each radial shell
bins = np.arange(0, r_max + bin_width, bin_width)  # Bin edges
pair_counts = np.zeros(len(bins) - 1)  # Initialize counts for each bin

# Calculate pair counts in each bin
for i in range(len(oxygen_positions)):
    for j in range(i + 1, len(oxygen_positions)):
        dist = np.linalg.norm(oxygen_positions[i] - oxygen_positions[j])
        if dist < r_max:
            bin_index = int(dist / bin_width)
            if bin_index < len(pair_counts):  # Ensure we are within bounds
                pair_counts[bin_index] += 1  # Increment the count for the␣
 ↪appropriate bin
```

```
# Calculate the volume for each shell
shell_volumes = 4 * np.pi * ((bins[1:] - bin_width / 2) ** 2) * bin_width  # 4 r² * Δr

# Calculate pairs per unit volume
pairs_per_volume = pair_counts / shell_volumes

# Calculate bin centers for plotting
bin_centers = bins[:-1] + bin_width / 2

# Plot number of pairs per unit volume vs radius
plt.figure(figsize=(10, 6))
plt.plot(bin_centers, pairs_per_volume, color='b', marker='o')
plt.xlabel('Radius (Å)')
plt.ylabel('Number of Pairs / Volume')
plt.title('Number of Pairs of Oxygen Atoms per Unit Volume vs Radius')
plt.grid(True)
plt.ylim(0, np.max(pairs_per_volume) * 1.1)  # Adjust y-axis for better visibility
plt.xlim(0, r_max)  # Set x-limits to show relevant range
plt.show()
```

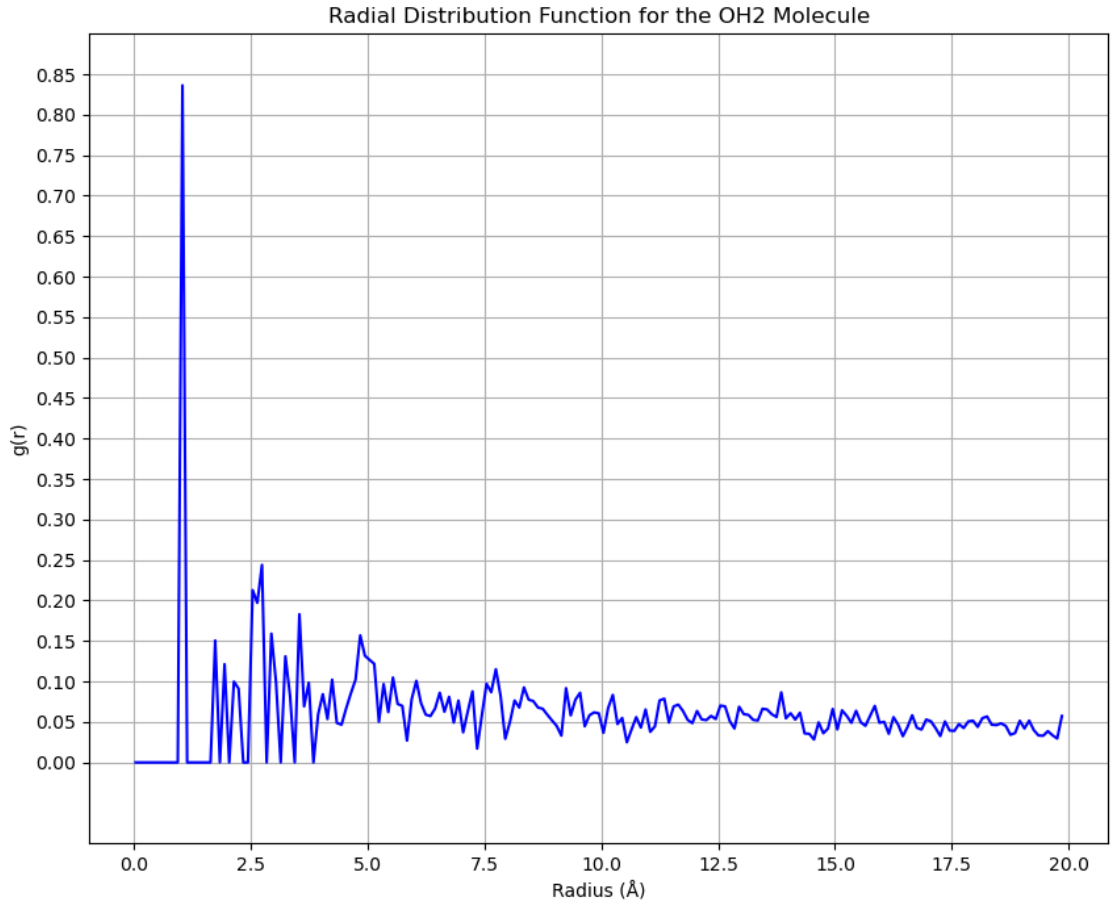


[24]:
```
# Find the index of the highest peak in g(r)
max_index = np.argmax(g_r)
peak_radius = bins[max_index] + bin_width / 2  # Center of the bin

# Define a small cutoff to consider atoms close to the peak radius
cutoff_distance = bin_width / 2  # You can adjust this if needed
lower_bound = peak_radius - cutoff_distance
upper_bound = peak_radius + cutoff_distance

# Count the number of pairs within the range of the highest peak
pairs_within_peak = 0

for i in range(len(oxygen_positions)):
    for j in range(i + 1, len(oxygen_positions)):
        dist = np.linalg.norm(oxygen_positions[i] - oxygen_positions[j])
        if lower_bound <= dist <= upper_bound:
            pairs_within_peak += 1  # Count each pair that falls within the␣
 ↪range

# Print the number of pairs contributing to the peak
print(f"Number of pairs contributing to the peak at {peak_radius:.2f} Å:␣
 ↪{pairs_within_peak}")
```

```
Number of pairs contributing to the peak at 1.10 Å: 2
```

[22]:
```
import numpy as np
import matplotlib.pyplot as plt

# Load your data (time frame, atom ID, x, y, z) - adjust the file path as needed
data = np.loadtxt('coord1.dat')
oxygen_positions = data[data[:, 1] % 3 == 1, 2:5]  # Select oxygen positions␣
 ↪assuming IDs modulo 3 are oxygen

# Parameters for g(r) calculation
r_max = 20.0  # Maximum radius to consider
bin_width = 0.1  # Width of each radial shell
bins = np.arange(0, r_max, bin_width)  # Bin edges
g_r = np.zeros(len(bins) - 1)

# Density estimation (number of particles per unit volume)
volume = (4/3) * np.pi * (r_max**3)
density = len(oxygen_positions) / volume

# Calculate g(r)
for i in range(len(oxygen_positions)):
    for j in range(i + 1, len(oxygen_positions)):
        dist = np.linalg.norm(oxygen_positions[i] - oxygen_positions[j])
```

```
        if dist < r_max:
            bin_index = int(dist / bin_width)
            if bin_index < len(g_r):  # Check to avoid out-of-bounds error
                g_r[bin_index] += 2  # Count each pair once

# Normalize g(r)
shell_volumes = (4/3) * np.pi * (np.diff(bins**3))  # Volume of each shell
g_r /= (shell_volumes * density * len(oxygen_positions))  # Normalize g(r)

# Rescale g(r) to a specific range, if needed
max_g_r = np.min([np.max(g_r), 2])  # Ensures g(r) does not exceed 2
g_r_scaled = np.clip(g_r, 0, max_g_r)

# Plot the radial distribution function
plt.figure(figsize=(10, 8))
plt.plot(bins[:-1] + bin_width / 2, g_r_scaled, color='b')  # Centered bin␣
 ↪edges for plotting
plt.xlabel('Radius (Å)')
plt.ylabel('g(r)')
plt.title('Radial Distribution Function for the OH2 Molecule')
plt.grid(True)

# Customize y-axis limits and ticks to focus on the range 0 to 1, with steps of␣
 ↪0.05
plt.ylim(-0.1, 0.9)  # Adjusted upper limit for better visibility
plt.yticks(np.arange(0, 0.9, 0.05))  # Ticks from 0 to 0.9 in steps of 0.05

plt.show()

# Additional code to check pairs within a specific distance
cutoff_distance = 3.0  # Example cutoff distance
pairs_within_cutoff = []

# Check all pairs of oxygen atoms within the cutoff distance
for i in range(len(oxygen_positions)):
    for j in range(i + 1, len(oxygen_positions)):
        dist = np.linalg.norm(oxygen_positions[i] - oxygen_positions[j])
        if np.isclose(dist, cutoff_distance, atol=0.01):  # Allow for small␣
 ↪floating-point errors
            pairs_within_cutoff.append((i, j, dist))
            print(f"Distance between atom {i} and atom {j}: {dist:.2f} Å")

# Print the total number of pairs within the cutoff distance
print(f"\nTotal number of pairs within {cutoff_distance} Å:␣
 ↪{len(pairs_within_cutoff)}")
```




```
Distance between atom 222 and atom 226: 2.99 Å

Total number of pairs within 3.0 Å: 1
```

[43]:
```
max_index = np.argmax(g_r)
max_radius = bins[max_index] + bin_width / 2  # Centered bin edge

# Print the index and corresponding radius
print(f"Index of max(g_r): {max_index}")
print(f"Radius at max(g_r): {max_radius:.2f} Å")
print(f"Max g(r) value: {g_r[max_index]:.2f}")
```

```
Index of max(g_r): 10
Radius at max(g_r): 1.05 Å
Max g(r) value: 2.88
```

[48]:
```
# pairwise distances between the first 20 oxygen atoms
first_20_oxygen = oxygen_positions[:100]
distances = []
```

```python
for i in range(len(first_20_oxygen)):
    for j in range(i + 1, len(first_20_oxygen)):
        dist = np.linalg.norm(first_20_oxygen[i] - first_20_oxygen[j])
        distances.append((i, j, dist))
        print(f"Distance between atom {i} and atom {j}: {dist:.2f} Å")
```

```
Distance between atom 0 and atom 1: 94.26 Å
Distance between atom 0 and atom 2: 84.06 Å
Distance between atom 0 and atom 3: 35.60 Å
Distance between atom 0 and atom 4: 30.18 Å
Distance between atom 0 and atom 5: 50.10 Å
Distance between atom 0 and atom 6: 85.40 Å
Distance between atom 0 and atom 7: 32.34 Å
Distance between atom 0 and atom 8: 82.47 Å
Distance between atom 0 and atom 9: 48.13 Å
Distance between atom 0 and atom 10: 86.62 Å
Distance between atom 0 and atom 11: 87.84 Å
Distance between atom 0 and atom 12: 81.97 Å
Distance between atom 0 and atom 13: 37.44 Å
Distance between atom 0 and atom 14: 95.65 Å
Distance between atom 0 and atom 15: 35.20 Å
Distance between atom 0 and atom 16: 40.66 Å
Distance between atom 0 and atom 17: 26.84 Å
Distance between atom 0 and atom 18: 89.42 Å
Distance between atom 0 and atom 19: 88.86 Å
Distance between atom 1 and atom 2: 24.31 Å
Distance between atom 1 and atom 3: 90.82 Å
Distance between atom 1 and atom 4: 118.31 Å
Distance between atom 1 and atom 5: 61.14 Å
Distance between atom 1 and atom 6: 49.56 Å
Distance between atom 1 and atom 7: 122.23 Å
Distance between atom 1 and atom 8: 17.85 Å
Distance between atom 1 and atom 9: 66.57 Å
Distance between atom 1 and atom 10: 27.92 Å
Distance between atom 1 and atom 11: 41.10 Å
Distance between atom 1 and atom 12: 82.02 Å
Distance between atom 1 and atom 13: 113.20 Å
Distance between atom 1 and atom 14: 24.47 Å
Distance between atom 1 and atom 15: 81.87 Å
Distance between atom 1 and atom 16: 62.90 Å
Distance between atom 1 and atom 17: 117.53 Å
Distance between atom 1 and atom 18: 35.81 Å
Distance between atom 1 and atom 19: 38.67 Å
Distance between atom 2 and atom 3: 79.89 Å
Distance between atom 2 and atom 4: 106.17 Å
Distance between atom 2 and atom 5: 51.02 Å
Distance between atom 2 and atom 6: 28.10 Å
```

```
Distance between atom 2 and atom 7: 109.83 Å
Distance between atom 2 and atom 8: 11.61 Å
Distance between atom 2 and atom 9: 61.92 Å
Distance between atom 2 and atom 10: 10.82 Å
Distance between atom 2 and atom 11: 26.48 Å
Distance between atom 2 and atom 12: 70.23 Å
Distance between atom 2 and atom 13: 96.04 Å
Distance between atom 2 and atom 14: 22.64 Å
Distance between atom 2 and atom 15: 73.23 Å
Distance between atom 2 and atom 16: 51.35 Å
Distance between atom 2 and atom 17: 104.82 Å
Distance between atom 2 and atom 18: 17.16 Å
Distance between atom 2 and atom 19: 17.77 Å
Distance between atom 3 and atom 4: 35.57 Å
Distance between atom 3 and atom 5: 30.99 Å
Distance between atom 3 and atom 6: 73.13 Å
Distance between atom 3 and atom 7: 40.63 Å
Distance between atom 3 and atom 8: 81.71 Å
Distance between atom 3 and atom 9: 29.80 Å
Distance between atom 3 and atom 10: 77.85 Å
Distance between atom 3 and atom 11: 92.91 Å
Distance between atom 3 and atom 12: 50.66 Å
Distance between atom 3 and atom 13: 45.53 Å
Distance between atom 3 and atom 14: 97.94 Å
Distance between atom 3 and atom 15: 11.61 Å
Distance between atom 3 and atom 16: 28.96 Å
Distance between atom 3 and atom 17: 39.80 Å
Distance between atom 3 and atom 18: 89.66 Å
Distance between atom 3 and atom 19: 87.99 Å
Distance between atom 4 and atom 5: 63.61 Å
Distance between atom 4 and atom 6: 100.70 Å
Distance between atom 4 and atom 7: 5.17 Å
Distance between atom 4 and atom 8: 106.77 Å
Distance between atom 4 and atom 9: 61.53 Å
Distance between atom 4 and atom 10: 106.48 Å
Distance between atom 4 and atom 11: 112.32 Å
Distance between atom 4 and atom 12: 83.67 Å
Distance between atom 4 and atom 13: 30.84 Å
Distance between atom 4 and atom 14: 121.00 Å
Distance between atom 4 and atom 15: 42.90 Å
Distance between atom 4 and atom 16: 57.22 Å
Distance between atom 4 and atom 17: 8.50 Å
Distance between atom 4 and atom 18: 112.22 Å
Distance between atom 4 and atom 19: 110.73 Å
Distance between atom 5 and atom 6: 47.22 Å
Distance between atom 5 and atom 7: 68.29 Å
Distance between atom 5 and atom 8: 53.40 Å
Distance between atom 5 and atom 9: 16.88 Å
```

```
Distance between atom 5 and atom 10: 48.00 Å
Distance between atom 5 and atom 11: 68.66 Å
Distance between atom 5 and atom 12: 38.84 Å
Distance between atom 5 and atom 13: 64.11 Å
Distance between atom 5 and atom 14: 70.37 Å
Distance between atom 5 and atom 15: 24.26 Å
Distance between atom 5 and atom 16: 11.12 Å
Distance between atom 5 and atom 17: 65.39 Å
Distance between atom 5 and atom 18: 63.36 Å
Distance between atom 5 and atom 19: 61.96 Å
Distance between atom 6 and atom 7: 104.43 Å
Distance between atom 6 and atom 8: 39.26 Å
Distance between atom 6 and atom 9: 62.67 Å
Distance between atom 6 and atom 10: 21.69 Å
Distance between atom 6 and atom 11: 44.34 Å
Distance between atom 6 and atom 12: 52.31 Å
Distance between atom 6 and atom 13: 87.24 Å
Distance between atom 6 and atom 14: 48.60 Å
Distance between atom 6 and atom 15: 69.96 Å
Distance between atom 6 and atom 16: 49.48 Å
Distance between atom 6 and atom 17: 100.17 Å
Distance between atom 6 and atom 18: 33.76 Å
Distance between atom 6 and atom 19: 29.87 Å
Distance between atom 7 and atom 8: 110.36 Å
Distance between atom 7 and atom 9: 66.42 Å
Distance between atom 7 and atom 10: 110.41 Å
Distance between atom 7 and atom 11: 115.04 Å
Distance between atom 7 and atom 12: 88.35 Å
Distance between atom 7 and atom 13: 30.46 Å
Distance between atom 7 and atom 14: 124.24 Å
Distance between atom 7 and atom 15: 47.90 Å
Distance between atom 7 and atom 16: 61.55 Å
Distance between atom 7 and atom 17: 7.06 Å
Distance between atom 7 and atom 18: 115.33 Å
Distance between atom 7 and atom 19: 113.84 Å
Distance between atom 8 and atom 9: 61.72 Å
Distance between atom 8 and atom 10: 20.76 Å
Distance between atom 8 and atom 11: 26.16 Å
Distance between atom 8 and atom 12: 77.31 Å
Distance between atom 8 and atom 13: 98.51 Å
Distance between atom 8 and atom 14: 17.45 Å
Distance between atom 8 and atom 15: 73.92 Å
Distance between atom 8 and atom 16: 52.79 Å
Distance between atom 8 and atom 17: 105.18 Å
Distance between atom 8 and atom 18: 21.54 Å
Distance between atom 8 and atom 19: 24.28 Å
Distance between atom 9 and atom 10: 59.69 Å
Distance between atom 9 and atom 11: 79.61 Å
```

```
Distance between atom 9 and atom 12: 47.80 Å
Distance between atom 9 and atom 13: 69.67 Å
Distance between atom 9 and atom 14: 78.96 Å
Distance between atom 9 and atom 15: 19.12 Å
Distance between atom 9 and atom 16: 21.08 Å
Distance between atom 9 and atom 17: 64.36 Å
Distance between atom 9 and atom 18: 75.32 Å
Distance between atom 9 and atom 19: 74.69 Å
Distance between atom 10 and atom 11: 36.15 Å
Distance between atom 10 and atom 12: 61.91 Å
Distance between atom 10 and atom 13: 97.00 Å
Distance between atom 10 and atom 14: 32.01 Å
Distance between atom 10 and atom 15: 71.58 Å
Distance between atom 10 and atom 16: 50.47 Å
Distance between atom 10 and atom 17: 105.88 Å
Distance between atom 10 and atom 18: 25.28 Å
Distance between atom 10 and atom 19: 24.31 Å
Distance between atom 11 and atom 12: 91.58 Å
Distance between atom 11 and atom 13: 97.55 Å
Distance between atom 11 and atom 14: 19.97 Å
Distance between atom 11 and atom 15: 87.27 Å
Distance between atom 11 and atom 16: 65.03 Å
Distance between atom 11 and atom 17: 109.01 Å
Distance between atom 11 and atom 18: 12.26 Å
Distance between atom 11 and atom 19: 15.45 Å
Distance between atom 12 and atom 13: 82.23 Å
Distance between atom 12 and atom 14: 92.72 Å
Distance between atom 12 and atom 15: 51.05 Å
Distance between atom 12 and atom 16: 47.85 Å
Distance between atom 12 and atom 17: 87.56 Å
Distance between atom 12 and atom 18: 82.46 Å
Distance between atom 12 and atom 19: 79.54 Å
Distance between atom 13 and atom 14: 110.35 Å
Distance between atom 13 and atom 15: 53.19 Å
Distance between atom 13 and atom 16: 56.01 Å
Distance between atom 13 and atom 17: 26.08 Å
Distance between atom 13 and atom 18: 98.05 Å
Distance between atom 13 and atom 19: 95.88 Å
Distance between atom 14 and atom 15: 90.50 Å
Distance between atom 14 and atom 16: 68.99 Å
Distance between atom 14 and atom 17: 118.64 Å
Distance between atom 14 and atom 18: 19.32 Å
Distance between atom 14 and atom 19: 23.91 Å
Distance between atom 15 and atom 16: 22.65 Å
Distance between atom 15 and atom 17: 46.32 Å
Distance between atom 15 and atom 18: 84.10 Å
Distance between atom 15 and atom 19: 82.92 Å
Distance between atom 16 and atom 17: 57.82 Å
```

```
Distance between atom 16 and atom 18: 61.52 Å
Distance between atom 16 and atom 19: 60.28 Å
Distance between atom 17 and atom 18: 109.72 Å
Distance between atom 17 and atom 19: 108.31 Å
Distance between atom 18 and atom 19: 4.79 Å
```

[18]:
```python
import numpy as np

# Load your data (time frame, atom ID, x, y, z) - adjust the file path as needed
data = np.loadtxt('coord1.dat')
oxygen_positions = data[data[:, 1] % 3 == 1, 2:5]  # Select oxygen positions␣
 ↪assuming IDs modulo 3 are oxygen

cutoff_distance = 3  # Set the cutoff distance to 11.61 Å
pairs_with_distance = []
tolerance = 0.01  # Allowable tolerance for floating-point comparison

# Check all pairs of oxygen atoms
for i in range(len(oxygen_positions)):
    for j in range(i + 1, len(oxygen_positions)):
        dist = np.linalg.norm(oxygen_positions[i] - oxygen_positions[j])
        # Check if the distance is within the specified tolerance
        if abs(dist - cutoff_distance) < tolerance:
            pairs_with_distance.append((i, j, dist))
            print(f"Distance between atom {i} and atom {j}: {dist:.2f} Å")

# Print the total number of pairs with a distance of approximately␣
 ↪cutoff_distance
print(f"\nTotal number of pairs with a distance of {cutoff_distance} Å:␣
 ↪{len(pairs_with_distance)}")
```

```
Distance between atom 222 and atom 226: 2.99 Å

Total number of pairs with a distance of 3 Å: 1
```

[46]:
```python
cutoff_distance = 1.1

for i in range(len(oxygen_positions)):
    for j in range(i + 1, len(oxygen_positions)):
        dist = np.linalg.norm(oxygen_positions[i] - oxygen_positions[j])
        if dist <= cutoff_distance:
            print(f"Close pair: Atom {i} and Atom {j} - Distance: {dist:.2f} Å")
```

```
Close pair: Atom 221 and Atom 286 - Distance: 1.03 Å
Close pair: Atom 299 and Atom 324 - Distance: 1.09 Å
```

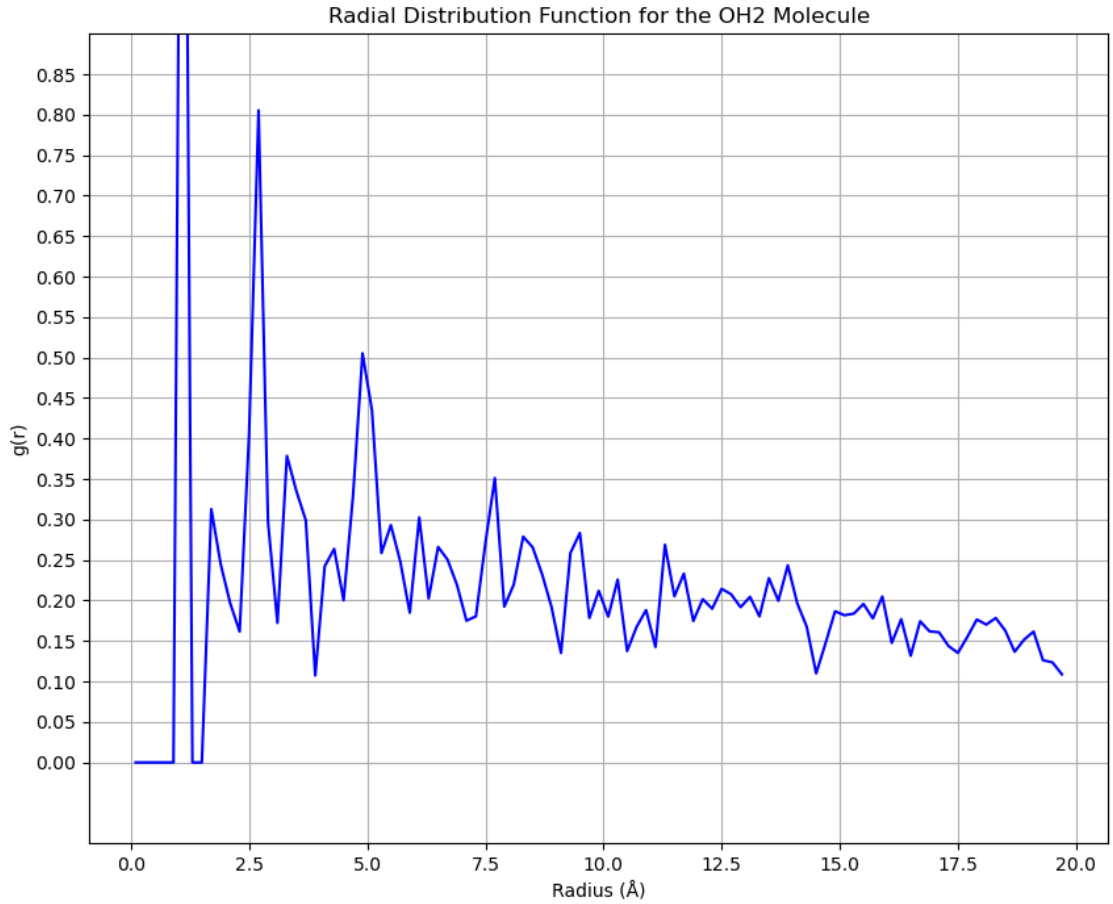
[19]:
```python
# Calculate g(r)
for i in range(len(oxygen_positions)):
```

```
    for j in range(i + 1, len(oxygen_positions)):
        dist = np.linalg.norm(oxygen_positions[i] - oxygen_positions[j])
        if dist < r_max:
            bin_index = int(dist / bin_width)
            if bin_index < len(g_r):  # Check to avoid out-of-bounds error
                g_r[bin_index] += 2  # Count each pair once

# Normalize g(r)
shell_volumes = (4/3) * np.pi * (np.diff(bins**3))
g_r /= (shell_volumes)

# Print bin edges and corresponding g(r) values
print("Bin (Radius Range) - g(r) value")
for i in range(len(g_r)):
    bin_start = bins[i]
    bin_end = bins[i] + bin_width
    print(f"{bin_start:.2f} Å - {bin_end:.2f} Å: g(r) = {g_r[i]:.3f}")

# Plot the radial distribution function
plt.figure(figsize=(10, 8))
plt.plot(bins[:-1] + bin_width / 2, g_r, color='b')  # Centered bin edges for␣
 ↪plotting
plt.xlabel('Radius (Å)')
plt.ylabel('g(r)')
plt.title('Radial Distribution Function for the OH2 Molecule')
plt.grid(True)

# Customize y-axis limits and ticks to focus on the range 0 to 1, with steps of␣
 ↪0.05
plt.ylim(-0.1, 0.9)  # Adjusted upper limit for better visibility
plt.yticks(np.arange(0, 0.9, 0.05))  # Ticks from 0 to 0.9 in steps of 0.05

plt.show()
```

```
Bin (Radius Range) - g(r) value
0.00 Å - 0.20 Å: g(r) = 0.000
0.20 Å - 0.40 Å: g(r) = 0.000
0.40 Å - 0.60 Å: g(r) = 0.000
0.60 Å - 0.80 Å: g(r) = 0.000
0.80 Å - 1.00 Å: g(r) = 0.000
1.00 Å - 1.20 Å: g(r) = 1.742
1.20 Å - 1.40 Å: g(r) = 0.000
1.40 Å - 1.60 Å: g(r) = 0.000
1.60 Å - 1.80 Å: g(r) = 0.313
1.80 Å - 2.00 Å: g(r) = 0.244
2.00 Å - 2.20 Å: g(r) = 0.197
2.20 Å - 2.40 Å: g(r) = 0.162
```

2.40 Å - 2.60 Å: g(r) = 0.406
2.60 Å - 2.80 Å: g(r) = 0.805
2.80 Å - 3.00 Å: g(r) = 0.297
3.00 Å - 3.20 Å: g(r) = 0.172
3.20 Å - 3.40 Å: g(r) = 0.379
3.40 Å - 3.60 Å: g(r) = 0.335
3.60 Å - 3.80 Å: g(r) = 0.299
3.80 Å - 4.00 Å: g(r) = 0.107
4.00 Å - 4.20 Å: g(r) = 0.242
4.20 Å - 4.40 Å: g(r) = 0.264
4.40 Å - 4.60 Å: g(r) = 0.200
4.60 Å - 4.80 Å: g(r) = 0.330
4.80 Å - 5.00 Å: g(r) = 0.505
5.00 Å - 5.20 Å: g(r) = 0.435
5.20 Å - 5.40 Å: g(r) = 0.259
5.40 Å - 5.60 Å: g(r) = 0.293
5.60 Å - 5.80 Å: g(r) = 0.248
5.80 Å - 6.00 Å: g(r) = 0.185
6.00 Å - 6.20 Å: g(r) = 0.303
6.20 Å - 6.40 Å: g(r) = 0.202
6.40 Å - 6.60 Å: g(r) = 0.266
6.60 Å - 6.80 Å: g(r) = 0.250
6.80 Å - 7.00 Å: g(r) = 0.219
7.00 Å - 7.20 Å: g(r) = 0.175
7.20 Å - 7.40 Å: g(r) = 0.181
7.40 Å - 7.60 Å: g(r) = 0.271
7.60 Å - 7.80 Å: g(r) = 0.351
7.80 Å - 8.00 Å: g(r) = 0.192
8.00 Å - 8.20 Å: g(r) = 0.220
8.20 Å - 8.40 Å: g(r) = 0.279
8.40 Å - 8.60 Å: g(r) = 0.266
8.60 Å - 8.80 Å: g(r) = 0.233
8.80 Å - 9.00 Å: g(r) = 0.192
9.00 Å - 9.20 Å: g(r) = 0.135
9.20 Å - 9.40 Å: g(r) = 0.259
9.40 Å - 9.60 Å: g(r) = 0.283
9.60 Å - 9.80 Å: g(r) = 0.178
9.80 Å - 10.00 Å: g(r) = 0.212
10.00 Å - 10.20 Å: g(r) = 0.180
10.20 Å - 10.40 Å: g(r) = 0.226
10.40 Å - 10.60 Å: g(r) = 0.138
10.60 Å - 10.80 Å: g(r) = 0.167
10.80 Å - 11.00 Å: g(r) = 0.188
11.00 Å - 11.20 Å: g(r) = 0.143
11.20 Å - 11.40 Å: g(r) = 0.269
11.40 Å - 11.60 Å: g(r) = 0.205
11.60 Å - 11.80 Å: g(r) = 0.233
11.80 Å - 12.00 Å: g(r) = 0.175

12.00 Å - 12.20 Å: g(r) = 0.202
12.20 Å - 12.40 Å: g(r) = 0.190
12.40 Å - 12.60 Å: g(r) = 0.214
12.60 Å - 12.80 Å: g(r) = 0.208
12.80 Å - 13.00 Å: g(r) = 0.192
13.00 Å - 13.20 Å: g(r) = 0.205
13.20 Å - 13.40 Å: g(r) = 0.180
13.40 Å - 13.60 Å: g(r) = 0.228
13.60 Å - 13.80 Å: g(r) = 0.200
13.80 Å - 14.00 Å: g(r) = 0.243
14.00 Å - 14.20 Å: g(r) = 0.197
14.20 Å - 14.40 Å: g(r) = 0.168
14.40 Å - 14.60 Å: g(r) = 0.110
14.60 Å - 14.80 Å: g(r) = 0.148
14.80 Å - 15.00 Å: g(r) = 0.187
15.00 Å - 15.20 Å: g(r) = 0.182
15.20 Å - 15.40 Å: g(r) = 0.184
15.40 Å - 15.60 Å: g(r) = 0.196
15.60 Å - 15.80 Å: g(r) = 0.178
15.80 Å - 16.00 Å: g(r) = 0.205
16.00 Å - 16.20 Å: g(r) = 0.148
16.20 Å - 16.40 Å: g(r) = 0.177
16.40 Å - 16.60 Å: g(r) = 0.132
16.60 Å - 16.80 Å: g(r) = 0.174
16.80 Å - 17.00 Å: g(r) = 0.162
17.00 Å - 17.20 Å: g(r) = 0.161
17.20 Å - 17.40 Å: g(r) = 0.144
17.40 Å - 17.60 Å: g(r) = 0.135
17.60 Å - 17.80 Å: g(r) = 0.155
17.80 Å - 18.00 Å: g(r) = 0.177
18.00 Å - 18.20 Å: g(r) = 0.170
18.20 Å - 18.40 Å: g(r) = 0.178
18.40 Å - 18.60 Å: g(r) = 0.163
18.60 Å - 18.80 Å: g(r) = 0.137
18.80 Å - 19.00 Å: g(r) = 0.152
19.00 Å - 19.20 Å: g(r) = 0.162
19.20 Å - 19.40 Å: g(r) = 0.126
19.40 Å - 19.60 Å: g(r) = 0.124
19.60 Å - 19.80 Å: g(r) = 0.109



[32]:
```
# Calculate g(r) and track pairwise distances in bins
for i in range(len(oxygen_positions)):
    for j in range(i + 1, len(oxygen_positions)):
        dist = np.linalg.norm(oxygen_positions[i] - oxygen_positions[j])
        if dist < r_max:
            bin_index = int(dist / bin_width)
            if bin_index < len(g_r):  # Check to avoid out-of-bounds error
                g_r[bin_index] += 2  # Count each pair once
                # Print pairwise distance and corresponding bin
                #print(f"Distance: {dist:.2f} Å, Bin Index: {bin_index}, Bin␣
 ↪Range: {bins[bin_index]:.2f} Å - {bins[bin_index] + bin_width:.2f} Å")

# Normalize g(r)
shell_volumes = (4/3) * np.pi * (np.diff(bins**3))
g_r /= (shell_volumes * density * len(oxygen_positions))

# Print final g(r) values for each bin
print("\nFinal g(r) values for each bin:")
```

```
for i in range(len(g_r)):
    bin_start = bins[i]
    bin_end = bins[i] + bin_width
    print(f"Radius Range {bin_start:.2f} Å - {bin_end:.2f} Å: g(r) = {g_r[i]:.
 ↪3f}")
```

```
Final g(r) values for each bin:
Radius Range 0.00 Å - 0.10 Å: g(r) = 0.000
Radius Range 0.10 Å - 0.20 Å: g(r) = 0.000
Radius Range 0.20 Å - 0.30 Å: g(r) = 0.000
Radius Range 0.30 Å - 0.40 Å: g(r) = 0.000
Radius Range 0.40 Å - 0.50 Å: g(r) = 0.000
Radius Range 0.50 Å - 0.60 Å: g(r) = 0.000
Radius Range 0.60 Å - 0.70 Å: g(r) = 0.000
Radius Range 0.70 Å - 0.80 Å: g(r) = 0.000
Radius Range 0.80 Å - 0.90 Å: g(r) = 0.000
Radius Range 0.90 Å - 1.00 Å: g(r) = 0.000
Radius Range 1.00 Å - 1.10 Å: g(r) = 1.055
Radius Range 1.10 Å - 1.20 Å: g(r) = 0.000
Radius Range 1.20 Å - 1.30 Å: g(r) = 0.000
Radius Range 1.30 Å - 1.40 Å: g(r) = 0.000
Radius Range 1.40 Å - 1.50 Å: g(r) = 0.000
Radius Range 1.50 Å - 1.60 Å: g(r) = 0.000
Radius Range 1.60 Å - 1.70 Å: g(r) = 0.000
Radius Range 1.70 Å - 1.80 Å: g(r) = 0.163
Radius Range 1.80 Å - 1.90 Å: g(r) = 0.000
Radius Range 1.90 Å - 2.00 Å: g(r) = 0.129
Radius Range 2.00 Å - 2.10 Å: g(r) = 0.000
Radius Range 2.10 Å - 2.20 Å: g(r) = 0.105
Radius Range 2.20 Å - 2.30 Å: g(r) = 0.095
Radius Range 2.30 Å - 2.40 Å: g(r) = 0.000
Radius Range 2.40 Å - 2.50 Å: g(r) = 0.000
Radius Range 2.50 Å - 2.60 Å: g(r) = 0.221
Radius Range 2.60 Å - 2.70 Å: g(r) = 0.204
Radius Range 2.70 Å - 2.80 Å: g(r) = 0.252
Radius Range 2.80 Å - 2.90 Å: g(r) = 0.000
Radius Range 2.90 Å - 3.00 Å: g(r) = 0.163
Radius Range 3.00 Å - 3.10 Å: g(r) = 0.102
Radius Range 3.10 Å - 3.20 Å: g(r) = 0.000
Radius Range 3.20 Å - 3.30 Å: g(r) = 0.134
Radius Range 3.30 Å - 3.40 Å: g(r) = 0.084
Radius Range 3.40 Å - 3.50 Å: g(r) = 0.000
Radius Range 3.50 Å - 3.60 Å: g(r) = 0.186
Radius Range 3.60 Å - 3.70 Å: g(r) = 0.070
Radius Range 3.70 Å - 3.80 Å: g(r) = 0.100
Radius Range 3.80 Å - 3.90 Å: g(r) = 0.000
Radius Range 3.90 Å - 4.00 Å: g(r) = 0.060
```

```
Radius Range 4.00 Å - 4.10 Å: g(r) = 0.086
Radius Range 4.10 Å - 4.20 Å: g(r) = 0.054
Radius Range 4.20 Å - 4.30 Å: g(r) = 0.103
Radius Range 4.30 Å - 4.40 Å: g(r) = 0.049
Radius Range 4.40 Å - 4.50 Å: g(r) = 0.047
Radius Range 4.50 Å - 4.60 Å: g(r) = 0.068
Radius Range 4.60 Å - 4.70 Å: g(r) = 0.086
Radius Range 4.70 Å - 4.80 Å: g(r) = 0.103
Radius Range 4.80 Å - 4.90 Å: g(r) = 0.158
Radius Range 4.90 Å - 5.00 Å: g(r) = 0.133
Radius Range 5.00 Å - 5.10 Å: g(r) = 0.128
Radius Range 5.10 Å - 5.20 Å: g(r) = 0.123
Radius Range 5.20 Å - 5.30 Å: g(r) = 0.051
Radius Range 5.30 Å - 5.40 Å: g(r) = 0.097
Radius Range 5.40 Å - 5.50 Å: g(r) = 0.063
Radius Range 5.50 Å - 5.60 Å: g(r) = 0.106
Radius Range 5.60 Å - 5.70 Å: g(r) = 0.073
Radius Range 5.70 Å - 5.80 Å: g(r) = 0.070
Radius Range 5.80 Å - 5.90 Å: g(r) = 0.027
Radius Range 5.90 Å - 6.00 Å: g(r) = 0.079
Radius Range 6.00 Å - 6.10 Å: g(r) = 0.101
Radius Range 6.10 Å - 6.20 Å: g(r) = 0.074
Radius Range 6.20 Å - 6.30 Å: g(r) = 0.059
Radius Range 6.30 Å - 6.40 Å: g(r) = 0.058
Radius Range 6.40 Å - 6.50 Å: g(r) = 0.067
Radius Range 6.50 Å - 6.60 Å: g(r) = 0.086
Radius Range 6.60 Å - 6.70 Å: g(r) = 0.063
Radius Range 6.70 Å - 6.80 Å: g(r) = 0.081
Radius Range 6.80 Å - 6.90 Å: g(r) = 0.049
Radius Range 6.90 Å - 7.00 Å: g(r) = 0.077
Radius Range 7.00 Å - 7.10 Å: g(r) = 0.037
Radius Range 7.10 Å - 7.20 Å: g(r) = 0.063
Radius Range 7.20 Å - 7.30 Å: g(r) = 0.088
Radius Range 7.30 Å - 7.40 Å: g(r) = 0.017
Radius Range 7.40 Å - 7.50 Å: g(r) = 0.058
Radius Range 7.50 Å - 7.60 Å: g(r) = 0.098
Radius Range 7.60 Å - 7.70 Å: g(r) = 0.087
Radius Range 7.70 Å - 7.80 Å: g(r) = 0.116
Radius Range 7.80 Å - 7.90 Å: g(r) = 0.083
Radius Range 7.90 Å - 8.00 Å: g(r) = 0.029
Radius Range 8.00 Å - 8.10 Å: g(r) = 0.050
Radius Range 8.10 Å - 8.20 Å: g(r) = 0.077
Radius Range 8.20 Å - 8.30 Å: g(r) = 0.068
Radius Range 8.30 Å - 8.40 Å: g(r) = 0.093
Radius Range 8.40 Å - 8.50 Å: g(r) = 0.078
Radius Range 8.50 Å - 8.60 Å: g(r) = 0.076
Radius Range 8.60 Å - 8.70 Å: g(r) = 0.068
Radius Range 8.70 Å - 8.80 Å: g(r) = 0.066
```

```
Radius Range 8.80 Å - 8.90 Å: g(r) = 0.059
Radius Range 8.90 Å - 9.00 Å: g(r) = 0.052
Radius Range 9.00 Å - 9.10 Å: g(r) = 0.045
Radius Range 9.10 Å - 9.20 Å: g(r) = 0.033
Radius Range 9.20 Å - 9.30 Å: g(r) = 0.092
Radius Range 9.30 Å - 9.40 Å: g(r) = 0.058
Radius Range 9.40 Å - 9.50 Å: g(r) = 0.078
Radius Range 9.50 Å - 9.60 Å: g(r) = 0.086
Radius Range 9.60 Å - 9.70 Å: g(r) = 0.045
Radius Range 9.70 Å - 9.80 Å: g(r) = 0.058
Radius Range 9.80 Å - 9.90 Å: g(r) = 0.062
Radius Range 9.90 Å - 10.00 Å: g(r) = 0.061
Radius Range 10.00 Å - 10.10 Å: g(r) = 0.037
Radius Range 10.10 Å - 10.20 Å: g(r) = 0.067
Radius Range 10.20 Å - 10.30 Å: g(r) = 0.084
Radius Range 10.30 Å - 10.40 Å: g(r) = 0.047
Radius Range 10.40 Å - 10.50 Å: g(r) = 0.055
Radius Range 10.50 Å - 10.60 Å: g(r) = 0.025
Radius Range 10.60 Å - 10.70 Å: g(r) = 0.041
Radius Range 10.70 Å - 10.80 Å: g(r) = 0.056
Radius Range 10.80 Å - 10.90 Å: g(r) = 0.043
Radius Range 10.90 Å - 11.00 Å: g(r) = 0.066
Radius Range 11.00 Å - 11.10 Å: g(r) = 0.038
Radius Range 11.10 Å - 11.20 Å: g(r) = 0.045
Radius Range 11.20 Å - 11.30 Å: g(r) = 0.077
Radius Range 11.30 Å - 11.40 Å: g(r) = 0.079
Radius Range 11.40 Å - 11.50 Å: g(r) = 0.049
Radius Range 11.50 Å - 11.60 Å: g(r) = 0.069
Radius Range 11.60 Å - 11.70 Å: g(r) = 0.072
Radius Range 11.70 Å - 11.80 Å: g(r) = 0.064
Radius Range 11.80 Å - 11.90 Å: g(r) = 0.053
Radius Range 11.90 Å - 12.00 Å: g(r) = 0.049
Radius Range 12.00 Å - 12.10 Å: g(r) = 0.064
Radius Range 12.10 Å - 12.20 Å: g(r) = 0.053
Radius Range 12.20 Å - 12.30 Å: g(r) = 0.052
Radius Range 12.30 Å - 12.40 Å: g(r) = 0.058
Radius Range 12.40 Å - 12.50 Å: g(r) = 0.054
Radius Range 12.50 Å - 12.60 Å: g(r) = 0.070
Radius Range 12.60 Å - 12.70 Å: g(r) = 0.069
Radius Range 12.70 Å - 12.80 Å: g(r) = 0.051
Radius Range 12.80 Å - 12.90 Å: g(r) = 0.042
Radius Range 12.90 Å - 13.00 Å: g(r) = 0.069
Radius Range 13.00 Å - 13.10 Å: g(r) = 0.060
Radius Range 13.10 Å - 13.20 Å: g(r) = 0.059
Radius Range 13.20 Å - 13.30 Å: g(r) = 0.053
Radius Range 13.30 Å - 13.40 Å: g(r) = 0.052
Radius Range 13.40 Å - 13.50 Å: g(r) = 0.066
Radius Range 13.50 Å - 13.60 Å: g(r) = 0.065
```

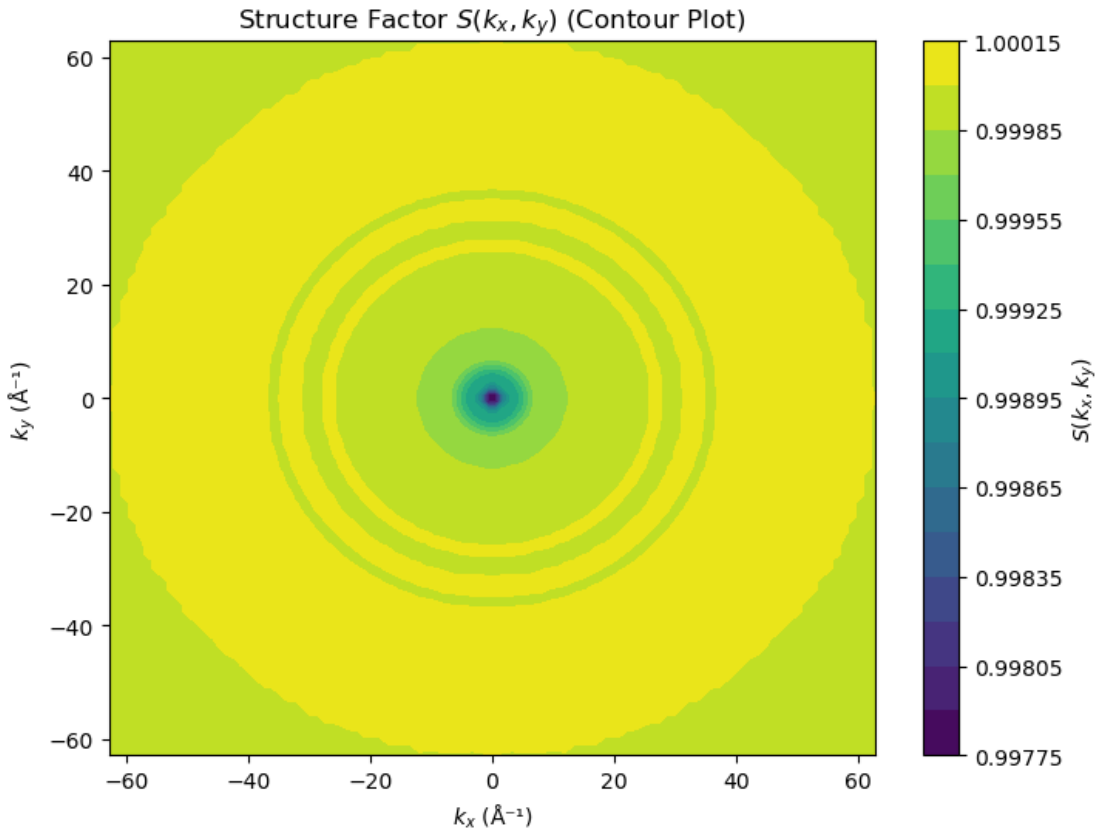
```
Radius Range 13.60 Å - 13.70 Å: g(r) = 0.060
Radius Range 13.70 Å - 13.80 Å: g(r) = 0.056
Radius Range 13.80 Å - 13.90 Å: g(r) = 0.087
Radius Range 13.90 Å - 14.00 Å: g(r) = 0.055
Radius Range 14.00 Å - 14.10 Å: g(r) = 0.061
Radius Range 14.10 Å - 14.20 Å: g(r) = 0.053
Radius Range 14.20 Å - 14.30 Å: g(r) = 0.061
Radius Range 14.30 Å - 14.40 Å: g(r) = 0.036
Radius Range 14.40 Å - 14.50 Å: g(r) = 0.035
Radius Range 14.50 Å - 14.60 Å: g(r) = 0.028
Radius Range 14.60 Å - 14.70 Å: g(r) = 0.049
Radius Range 14.70 Å - 14.80 Å: g(r) = 0.036
Radius Range 14.80 Å - 14.90 Å: g(r) = 0.042
Radius Range 14.90 Å - 15.00 Å: g(r) = 0.066
Radius Range 15.00 Å - 15.10 Å: g(r) = 0.041
Radius Range 15.10 Å - 15.20 Å: g(r) = 0.064
Radius Range 15.20 Å - 15.30 Å: g(r) = 0.058
Radius Range 15.30 Å - 15.40 Å: g(r) = 0.049
Radius Range 15.40 Å - 15.50 Å: g(r) = 0.064
Radius Range 15.50 Å - 15.60 Å: g(r) = 0.050
Radius Range 15.60 Å - 15.70 Å: g(r) = 0.045
Radius Range 15.70 Å - 15.80 Å: g(r) = 0.058
Radius Range 15.80 Å - 15.90 Å: g(r) = 0.070
Radius Range 15.90 Å - 16.00 Å: g(r) = 0.049
Radius Range 16.00 Å - 16.10 Å: g(r) = 0.050
Radius Range 16.10 Å - 16.20 Å: g(r) = 0.035
Radius Range 16.20 Å - 16.30 Å: g(r) = 0.056
Radius Range 16.30 Å - 16.40 Å: g(r) = 0.047
Radius Range 16.40 Å - 16.50 Å: g(r) = 0.032
Radius Range 16.50 Å - 16.60 Å: g(r) = 0.044
Radius Range 16.60 Å - 16.70 Å: g(r) = 0.058
Radius Range 16.70 Å - 16.80 Å: g(r) = 0.043
Radius Range 16.80 Å - 16.90 Å: g(r) = 0.041
Radius Range 16.90 Å - 17.00 Å: g(r) = 0.053
Radius Range 17.00 Å - 17.10 Å: g(r) = 0.051
Radius Range 17.10 Å - 17.20 Å: g(r) = 0.042
Radius Range 17.20 Å - 17.30 Å: g(r) = 0.033
Radius Range 17.30 Å - 17.40 Å: g(r) = 0.051
Radius Range 17.40 Å - 17.50 Å: g(r) = 0.039
Radius Range 17.50 Å - 17.60 Å: g(r) = 0.039
Radius Range 17.60 Å - 17.70 Å: g(r) = 0.047
Radius Range 17.70 Å - 17.80 Å: g(r) = 0.042
Radius Range 17.80 Å - 17.90 Å: g(r) = 0.051
Radius Range 17.90 Å - 18.00 Å: g(r) = 0.052
Radius Range 18.00 Å - 18.10 Å: g(r) = 0.044
Radius Range 18.10 Å - 18.20 Å: g(r) = 0.055
Radius Range 18.20 Å - 18.30 Å: g(r) = 0.057
Radius Range 18.30 Å - 18.40 Å: g(r) = 0.047
```

```
Radius Range 18.40 Å - 18.50 Å: g(r) = 0.046
Radius Range 18.50 Å - 18.60 Å: g(r) = 0.048
Radius Range 18.60 Å - 18.70 Å: g(r) = 0.045
Radius Range 18.70 Å - 18.80 Å: g(r) = 0.034
Radius Range 18.80 Å - 18.90 Å: g(r) = 0.036
Radius Range 18.90 Å - 19.00 Å: g(r) = 0.051
Radius Range 19.00 Å - 19.10 Å: g(r) = 0.042
Radius Range 19.10 Å - 19.20 Å: g(r) = 0.052
Radius Range 19.20 Å - 19.30 Å: g(r) = 0.040
Radius Range 19.30 Å - 19.40 Å: g(r) = 0.033
Radius Range 19.40 Å - 19.50 Å: g(r) = 0.033
Radius Range 19.50 Å - 19.60 Å: g(r) = 0.039
Radius Range 19.60 Å - 19.70 Å: g(r) = 0.033
Radius Range 19.70 Å - 19.80 Å: g(r) = 0.030
Radius Range 19.80 Å - 19.90 Å: g(r) = 0.057
```

[8]:
```python
import numpy as np
import matplotlib.pyplot as plt

# Load your data (time frame, atom ID, x, y, z) - adjust the file path as needed
data = np.loadtxt('coord1.dat')
oxygen_positions = data[data[:, 1] % 3 == 1, 2:5]  # Select oxygen positions␣
 ↪assuming IDs modulo 3 are oxygen

# Parameters for g(r) calculation
r_max = 35.0  # Maximum radius to consider
bin_width = 0.1  # Width of each radial shell
bins = np.arange(0, r_max, bin_width)  # Bin edges
g_r = np.zeros(len(bins) - 1)

# Calculate the density using only the volume within r_max
num_oxygen = len(oxygen_positions)
volume = (4/3) * np.pi * (r_max**3)
density = num_oxygen / volume

# Calculate g(r) by counting neighbors in spherical shells
for i in range(num_oxygen):
    for j in range(i + 1, num_oxygen):
        dist = np.linalg.norm(oxygen_positions[i] - oxygen_positions[j])
        if dist < r_max:
            bin_index = int(dist / bin_width)
            if bin_index < len(g_r):  # Check to avoid out-of-bounds error
                g_r[bin_index] += 2  # Count each pair once

# Normalize g(r) by shell volumes and density
shell_volumes = (4/3) * np.pi * ((bins[1:]**3) - (bins[:-1]**3))  # Volume of␣
 ↪each shell
```

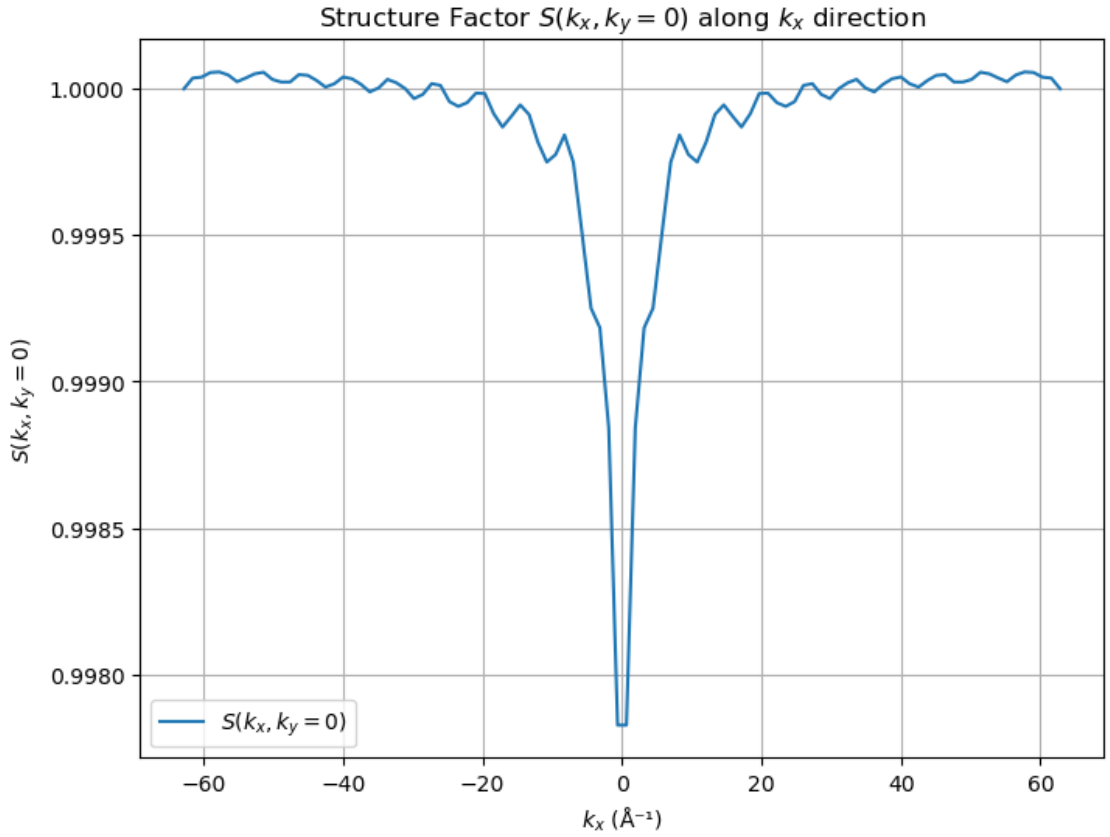
```
g_r /= (shell_volumes * density * num_oxygen)

# Define a 2D grid of k_x and k_y values
k_max = 2 * np.pi / bin_width  # Max k based on the bin width
k_points = 100  # Resolution of k grid
k_x_values = np.linspace(-k_max, k_max, k_points)
k_y_values = np.linspace(-k_max, k_max, k_points)
k_x, k_y = np.meshgrid(k_x_values, k_y_values)
k_values = np.sqrt(k_x**2 + k_y**2)  # Calculate radial k for each point on the␣
 ↪grid

# Calculate S(k_x, k_y) using the radial distribution function g(r)
S_k = np.zeros_like(k_values)

for i in range(k_points):
    for j in range(k_points):
        k = k_values[i, j]
        if k != 0:
            integrand = (g_r - 1) * np.sin(k * bins[:-1]) / (k * bins[:-1])
            integrand[0] = 0  # Avoid division by zero at the origin
            # Perform numerical integration using the trapezoidal rule
            S_k[i, j] = 1 + density * np.trapz(integrand, bins[:-1])

# Plot S(k_x, k_y) as a contour plot
plt.figure(figsize=(8, 6))
contour = plt.contourf(k_x, k_y, S_k, levels=20, cmap='viridis')
plt.colorbar(contour, label='$S(k_x, k_y)$')
plt.xlabel('$k_x$ (Å$^{-1}$)')
plt.ylabel('$k_y$ (Å$^{-1}$)')
plt.title('Structure Factor $S(k_x, k_y)$ (Contour Plot)')
plt.show()
```

```
C:\Users\Katha\AppData\Local\Temp\ipykernel_8772\2859220095.py:47:
RuntimeWarning: invalid value encountered in divide
  integrand = (g_r - 1) * np.sin(k * bins[:-1]) / (k * bins[:-1])
```



[9]:
```
import numpy as np
import matplotlib.pyplot as plt

# Assuming bin_width is already defined
k_max = 2 * np.pi / bin_width  # Max k based on the bin width
k_points = 100  # Resolution of k grid
k_x_values = np.linspace(-k_max, k_max, k_points)
k_y_values = np.linspace(-k_max, k_max, k_points)
k_x, k_y = np.meshgrid(k_x_values, k_y_values)
k_values = np.sqrt(k_x**2 + k_y**2)  # Calculate radial k for each point on the␣
↪grid

ky_zero_index = np.argmin(np.abs(k_y_values))

# Extract the slice of S(k_x, k_y) at k_y = 0
S_k_slice = S_k[:, ky_zero_index]

# Plot the slice along the k_x direction
plt.figure(figsize=(8, 6))
plt.plot(k_x_values, S_k_slice, label='$S(k_x, k_y=0)$')
```

```
plt.xlabel('$k_x$ (Å ⁻¹)')
plt.ylabel('$S(k_x, k_y=0)$')
plt.title('Structure Factor $S(k_x, k_y=0)$ along $k_x$ direction')
plt.legend()
plt.grid(True)
plt.show()
```

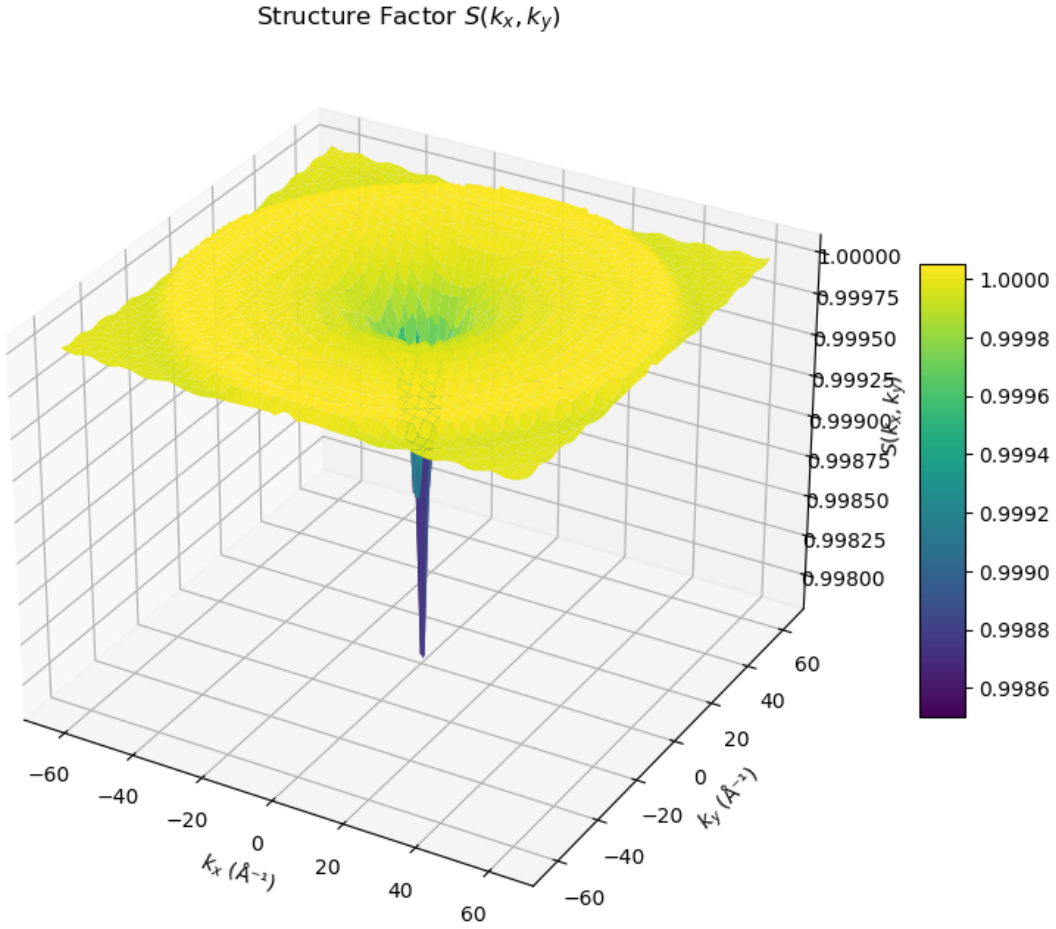


[106]:
```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Load your data (time frame, atom ID, x, y, z) - adjust the file path as needed
data = np.loadtxt('coord1.dat')
oxygen_positions = data[data[:, 1] % 3 == 1, 2:5]  # Select oxygen positions␣
 ↪assuming IDs modulo 3 are oxygen

# Parameters for g(r) calculation
r_max = 35.0  # Maximum radius to consider
bin_width = 0.1  # Width of each radial shell
bins = np.arange(0, r_max, bin_width)  # Bin edges
```

```
g_r = np.zeros(len(bins) - 1)

# Calculate the density using only the volume within r_max
num_oxygen = len(oxygen_positions)
volume = (4/3) * np.pi * (r_max**3)
density = num_oxygen / volume

# Calculate g(r) by counting neighbors in spherical shells
for i in range(num_oxygen):
    for j in range(i + 1, num_oxygen):
        dist = np.linalg.norm(oxygen_positions[i] - oxygen_positions[j])
        if dist < r_max:
            bin_index = int(dist / bin_width)
            if bin_index < len(g_r):  # Check to avoid out-of-bounds error
                g_r[bin_index] += 2  # Count each pair once

# Normalize g(r) by shell volumes and density
shell_volumes = (4/3) * np.pi * ((bins[1:]**3) - (bins[:-1]**3))  # Volume of␣
 ↪each shell
g_r /= (shell_volumes * density * num_oxygen)

# Define a 2D grid of k_x and k_y values
k_max = 2 * np.pi / bin_width  # Max k based on the bin width
k_points = 100  # Resolution of k grid
k_x_values = np.linspace(-k_max, k_max, k_points)
k_y_values = np.linspace(-k_max, k_max, k_points)
k_x, k_y = np.meshgrid(k_x_values, k_y_values)
k_values = np.sqrt(k_x**2 + k_y**2)  # Calculate radial k for each point on the␣
 ↪grid

# Calculate S(k_x, k_y) using the radial distribution function g(r)
S_k = np.zeros_like(k_values)

for i in range(k_points):
    for j in range(k_points):
        k = k_values[i, j]
        if k != 0:
            integrand = (g_r - 1) * np.sin(k * bins[:-1]) / (k * bins[:-1])
            integrand[0] = 0  # Avoid division by zero at the origin
            # Perform numerical integration using the trapezoidal rule
            S_k[i, j] = 1 + density * np.trapz(integrand, bins[:-1])

# Plot S(k_x, k_y) as a surface plot
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')
surf = ax.plot_surface(k_x, k_y, S_k, cmap='viridis', edgecolor='none')
ax.set_xlabel('$k_x$ (Å ⁻¹)')
```

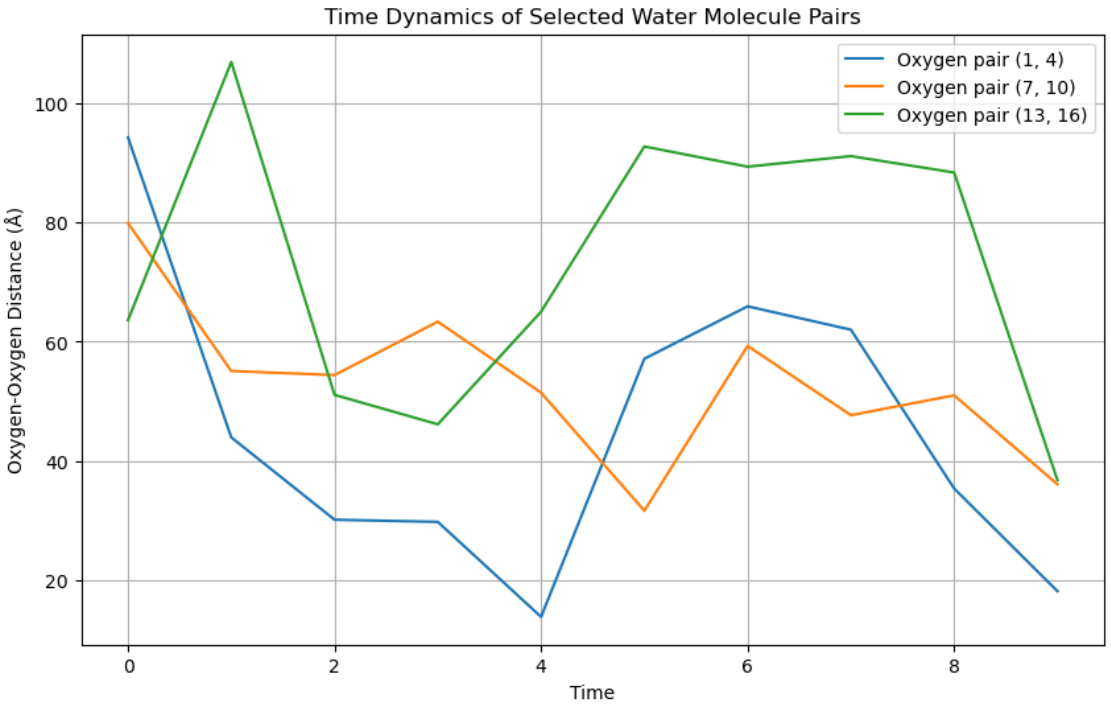
```
ax.set_ylabel('$k_y$ (Å ⁻¹)')
ax.set_zlabel('$S(k_x, k_y)$')
ax.set_title('Structure Factor $S(k_x, k_y)$')
fig.colorbar(surf, ax=ax, shrink=0.5, aspect=10)
plt.show()
```

```
C:\Users\Katha\AppData\Local\Temp\ipykernel_7824\4152695198.py:48:
RuntimeWarning: invalid value encountered in divide
  integrand = (g_r - 1) * np.sin(k * bins[:-1]) / (k * bins[:-1])
```



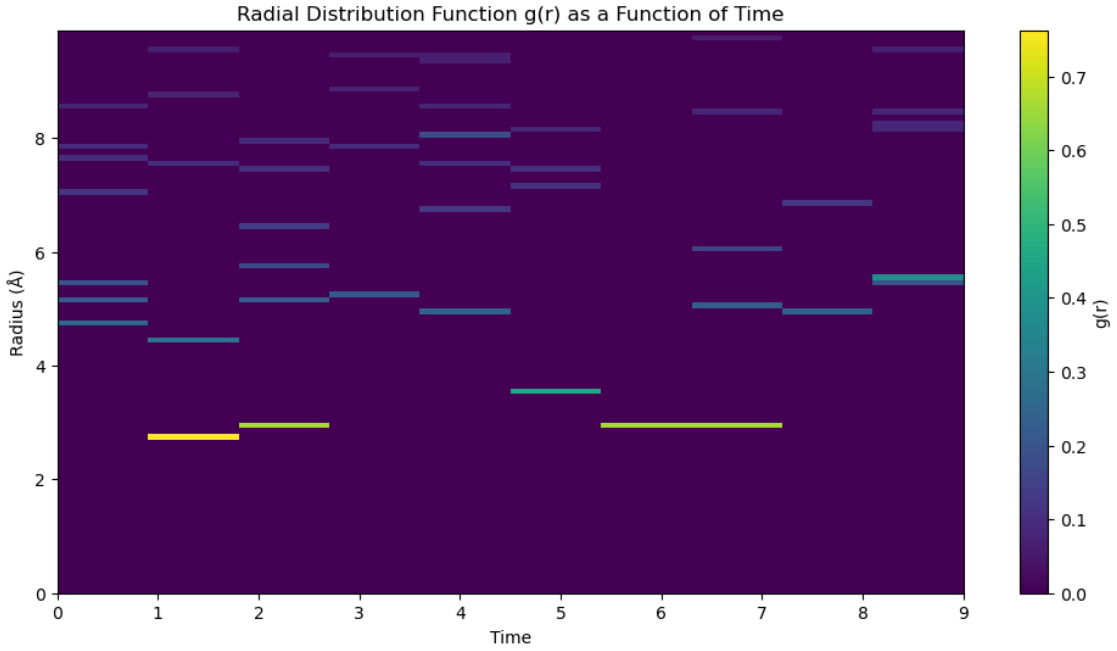
[107]:
```
import numpy as np
import matplotlib.pyplot as plt

# Load your data (assuming the file structure: time frame, atom ID, x, y, z)
data = np.loadtxt('coord1.dat')

# Extract unique time frames from the data
```

```
time_frames = np.unique(data[:, 0])

# Define pairs of water molecules (by their atom IDs) to track their␣
 ↪oxygen-oxygen distances
# Here we assume IDs of water oxygen atoms as 1, 4, 7, etc. Adjust the pairs as␣
 ↪needed
pairs = [(1, 4), (7, 10), (13, 16)]  # Example pairs

# Initialize a dictionary to store distances for each pair across all time␣
 ↪frames
pair_distances = {pair: [] for pair in pairs}

# Iterate over each time frame and calculate distances between specified pairs
for time in time_frames:
    # Extract positions for the current time frame
    frame_data = data[data[:, 0] == time]

    for pair in pairs:
        atom1_data = frame_data[frame_data[:, 1] == pair[0], 2:5]  # Position␣
 ↪of first atom in pair
        atom2_data = frame_data[frame_data[:, 1] == pair[1], 2:5]  # Position␣
 ↪of second atom in pair

        if len(atom1_data) > 0 and len(atom2_data) > 0:
            # Calculate distance between the two atoms
            distance = np.linalg.norm(atom1_data - atom2_data)
            pair_distances[pair].append(distance)

# Plot the distances over time
plt.figure(figsize=(10, 6))
for pair, distances in pair_distances.items():
    plt.plot(time_frames, distances, label=f'Oxygen pair {pair}')

plt.xlabel('Time')
plt.ylabel('Oxygen-Oxygen Distance (Å)')
plt.title('Time Dynamics of Selected Water Molecule Pairs')
plt.legend()
plt.grid(True)
plt.show()
```



[109]:
```python
import numpy as np
import matplotlib.pyplot as plt

# Load your data (assuming the file structure: time frame, atom ID, x, y, z)
data = np.loadtxt('coord1.dat')

# Parameters for g(r) calculation
r_max = 10.0  # Maximum radius to consider
bin_width = 0.1  # Width of each radial shell
bins = np.arange(0, r_max, bin_width)  # Bin edges
num_bins = len(bins) - 1

# Extract unique time frames from the data
time_frames = np.unique(data[:, 0])

# Identify oxygen atoms (assuming oxygen atoms are identified as IDs % 3 == 1)
oxygen_data = data[data[:, 1] % 3 == 1]

# Number of oxygen atoms (water molecules)
num_oxygen = len(np.unique(oxygen_data[:, 1]))

# Initialize an array to store g(r) for each time frame
g_r_time = np.zeros((len(time_frames), num_bins))
```

```
# Calculate the density using only the volume within r_max
volume = (4/3) * np.pi * (r_max**3)
density = num_oxygen / volume

# Loop over each time frame
for t_idx, time in enumerate(time_frames):
    # Extract positions of oxygen atoms for the current time frame
    frame_data = oxygen_data[oxygen_data[:, 0] == time]
    positions = frame_data[:, 2:5]  # Only x, y, z columns

    # Temporary g(r) for this time frame
    g_r = np.zeros(num_bins)

    # Calculate distances between all pairs and fill g(r) histogram
    for i in range(num_oxygen):
        for j in range(i + 1, num_oxygen):
            dist = np.linalg.norm(positions[i] - positions[j])
            if dist < r_max:
                bin_index = int(dist / bin_width)
                if bin_index < num_bins:
                    g_r[bin_index] += 2  # Count each pair once

    # Normalize g(r) by shell volumes and density
    shell_volumes = (4/3) * np.pi * ((bins[1:]**3) - (bins[:-1]**3))
    g_r /= (shell_volumes * density * num_oxygen)

    # Store this frame's g(r) in the array
    g_r_time[t_idx, :] = g_r

# Plot g(r) as a function of time as a heatmap
plt.figure(figsize=(12, 6))
plt.imshow(g_r_time.T, aspect='auto', origin='lower', extent=[time_frames[0],␣
 ↪time_frames[-1], bins[0], bins[-1]], cmap='viridis')
plt.colorbar(label='g(r)')
plt.xlabel('Time')
plt.ylabel('Radius (Å)')
plt.title('Radial Distribution Function g(r) as a Function of Time')
plt.show()
```


Radial Distribution Function g(r) as a Function of Time
Radius (Å)
Time
g(r)
0
2
4
6
8
0
1
2
3
4
5
6
7
8
9
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7

[ ]: