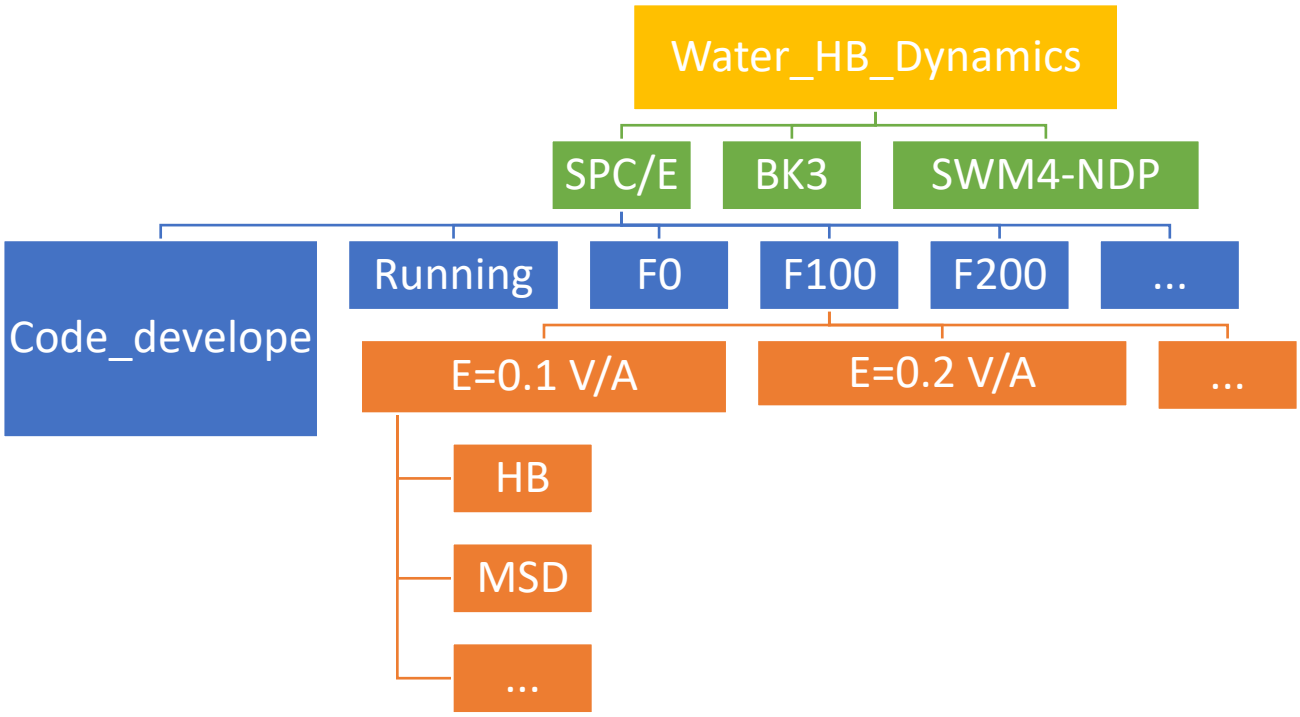
Appendix 6. Coding

1. Introduction

In this section we explain how the computations have been done. All the simulations, codes, and scripts are stored in a folder named “Water_HB_Dynamics” in one of the storages in Luzar lab called *“mondo”*. The trajectories would also be stored, but they are reproducible using the embedded input scripts and the corresponding simulation package: Lammps or Gromacs. The installation of the packages are also stored in the main folder. Here is how the files are arranged:

2. Folder format



For each water model there is a code-develop folder, where the required codes have been developed. There is also a *running* folder, where the running scripts are developed there.

The running scripts are the scripts that automatically run simulations, do calculations, or print out a set of results on a series of the systems. For instance, if we want to calculate the diffusion coefficient of all SPC/E water systems, we run a script that calculates the

slope of the MSD plot for all the frequencies and electric fields and writes the results in a file.

Each water model has a **"F0"** folder, means zero frequency E-field, or DC field. In the rest of the folders, the number in front of F gives the frequency of the E-field, for example all the systems in the F100 folder are water under the frequency of 100GHz.

In each frequency folder, there are electric field folders, and the value of the E-field in terms of the $V/\text{Å}$ is written in front of "**ef**" in the folder name. For instance, the system in the F100/ef.10 contains water under AC field with 100GHz frequency and $E_0 = 0.1\ V/\text{Å}$. For each system, we calculate several quantities, for each quantity we have a sub-folder.

3. Inverse Laplace transform calculation

```
// code Inverse Laplace transform for Luzar second part of
model

  // this procedure does the INVERES LAPLACE TRANSFORM based on
stehfest method, page 144, eq 7.7 of book Numerical Methods for
Laplace Transform Inversion, Alan M. Cohen
  // other important reference is : ref12 of the book,
Performance of numerical inversion of Laplace transforms,
Advances in Engineering Software and Workstations Volume 13,
Issue 3, May 1991, Pages 148â€"155


  // this program does the ILT of function 6, which is eq 13 in
Luzar conundrum paper.

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <iostream>
#include <fstream>
#include <vector>
#include <algorithm>   // for transform...
#include <functional>   // std::plus
#include <numeric>      // std::accumulate
```

```
using namespace std;

double func1(double func_input);
double func2(double func_input);
double func3(double func_input1,double func_input2);

double func4(double func_input1);
double func5(double func_input1);

unsigned int factorial(unsigned int n);


double integral(double intg_start, double intg_end, double dx, double
(*functn) (double), double s );
double integral2(double intg_start, double intg_end, double dx,
double (*functn) (double,double), double t );

double integral_laplace(double intg_start, double intg_end, double
dx, double (*functn) (double), double s );
double integral_INV_laplace(double intg_start, double intg_end,
double dx, double (*functn) (double), double t );

double K(int n, int N); // this functions computes K_n in second line
of eq 7.7 chapter 7 of the book Numerical Methods for Laplace
Transform Inversion, Alan M. Cohen
                        // N is arbirtrary even number and is number
of terms to do the expansion

double func6(double s, double k_f, double k_b, double tauD);
double func7(double s, double k_f, double k_b, double tauD);


#define PI 3.14159265
#define ln2 0.693147181

main(int argc, char *argv[]){

  double k_f=0.38, k_b=0.78, Diffusion=2.55, a0=1.5,
alpha=sqrt(10)/pow(6*PI*PI,0.333333),
tauD=(alpha*alpha*a0*a0)/Diffusion;  // alpha is in the formula,
sqrt(10) is for converting the units

// The units have been changing so that: k_b and k_f are in ps^-1,
Diffusion in x10^-9 x m / s^2, a0 is the first shell radius in
angstrom

   if(argc!=4){

   printf("Calculating Inverse Laplace transform of Luzar
equation\n");
   printf("Usage: ./ILT k_f k_b tauD\n");
```

```
   }
   else{

     argv++;  //The first argument of the main is useless

     for(int i=1; i<4;i++){
        char temp[50];
      strcpy(temp,*argv++);
      if(i==1){k_f=atof(temp);  if(k_f>0.0){printf("k_f=%lf \n",
k_f);} else{printf("Error, K_f must be a number greater than 0.0\n");
exit(1); } }
      if(i==2){k_b=atof(temp);  if(k_b>0.0){printf("k_b=%lf \n",
k_b);} else{printf("Error, k_b must be a number greater than 0.0\n");
exit(1); } }
      if(i==3){tauD=atof(temp);  if(tauD>0.0){printf("tauD=%lf \n",
tauD);} else{printf("Error, tauD must be a number greater than
0.0\n"); exit(1); } }
     }


   }


  // strcpy(dumpname,*argv++);

   char resultname[50];
   sprintf(resultname, "Inv_laplace_kb%2.2f_kf%2.2f_tauD%2.2f.dat",
k_b, k_f, tauD);

   printf("result name is %s\n",resultname);


  FILE *laplace_file, *inv_laplace_file;



  laplace_file=fopen("laplace_transform.dat", "w");
  inv_laplace_file=fopen(resultname, "w");



  int N=10;
  double t=0.1;




  while(t<30){
```

```
    if(t-floor(t)<0.00001){
  printf(" t %lf \n", t);
  }


    double a=ln2/t, kt=0, k_in_t=0;

    for(int n=1; n<=N;n++){ // the main summation in eq 7.7 of the
book

       // printf("n=%i \n",n);
     //ft+=(K(n,N)*func5(a*n));
        // Hint! a0 is a in the formula in luzar model:
     kt+=( K(n,N)*func6(a*n, k_f, k_b, tauD) );
     k_in_t+=( K(n,N)*func7(a*n, k_f, k_b, tauD) );

    }
    fprintf(inv_laplace_file, "%lf %lf %lf\n", t, a*kt , a*k_in_t );
    t+=0.01;
  }




}

double func6(double s, double k_f, double k_b, double tauD){ //func6
is f(s),
    return k_f/( s+k_f+3*k_b*s*tauD*( 1.0-sqrt(s*tauD)*atan(
1.0/(sqrt(s*tauD) ) ) ) );
}

double func7(double s, double k_f, double k_b, double tauD){ //func6
is f(s),
    return (k_f/k_b)*(1-(s+k_f)/( s+k_f+3*k_b*s*tauD*( 1.0-
sqrt(s*tauD)*atan( 1.0/(sqrt(s*tauD) ) ) ) ) );
}


double K(int n, int N){   // Hint! K_n is independent of the
function!

      int coef1=1;   // coef1 is (-1)^n+N/2
      if((n+N/2)%2==1)coef1=-1;

      double sum=0.0;

      for(int k=int((n+1)/2); k<=min(n,N/2); k++){

       //printf(" k=%i \n",k);
```

```
        int coef2=1;  // coef2 is (2k)! / (2k-n)!  Hint! in the main
formula it is 2k! in nominator, but a (2k-n)! in denominator
        for(int l=2*k; l>(2*k-n); l--){coef2*=l;}

        int fact_k_1=factorial(k-1);  // I seperated this because it
needs to go to power 2. In the formula we have k! (k-1)!= k (k-1)!
(k-1)! = k ( (k-1)! ) ^2

        sum+=( (pow(k,N/2)*coef2) / (factorial(N/2-
k)*k*fact_k_1*fact_k_1*factorial(n-k)) );
      }

    return coef1*sum;

}


unsigned int factorial(unsigned int n)
{
    if (n == 0)
       return 1;
    return n * factorial(n - 1);
}
```

4. Running scripts

Below is one example of bash script which runs the “CorrCoss9” code to calculate the re-orientation of water molecules in Laage model:

```
folderRunning=$(pwd)

cd ..

folderBase=$(pwd)


for f in 0  # 100 200 500  # Loop On the frequencies
do

if [ $f -eq 0 ]  Different Frequencies may need different E-field range
then
fieldRange="0 5 50  100 160  200 "
fi


folderf="$folderBase/F$f"

echo "fields that are started" >  $folderf/runned_CorrCoss
```

```
echo "folder f is $folderf"

for i in $fieldRange
do
   k=$(echo "$i/1000.0" | bc -l)
   echo "Welcome $i "
pwd


folderEfield="$folderf/ef$k"


#*********** CorrCss  ************
folderMSDMSR="$folderEfield/CorrCoss9"

mkdir $folderMSDMSR

cp $folderRunning/corrCoss_9  $folderMSDMSR

echo $folderMSDMSR


     #**** step 1 making the setup file:

echo "  Sometimes we need a script inside a script

# timestep 1.0

# efield 0.0 0.0 0.0.

# dump is writte every 10

# msd_step  100
# msd_lenghth  2000
# acv_lenghth  100
# hb_corrolation_lenghth 1500
# c_p 30
# c_m 2
# c_s 11
# print_step 100

# static properties:
calculateStep 10

# oo_max 3.5
# c_min 0.86602
# max_r_o_h 2.4



# For PDB SPCE:
# identifier TIMESTEP
# useless_words 15
# atom_in_mol 3
# numberOfAtoms 1536
# scaled 2  (1 for scaled)

" > $folderMSDMSR/setup_file
```

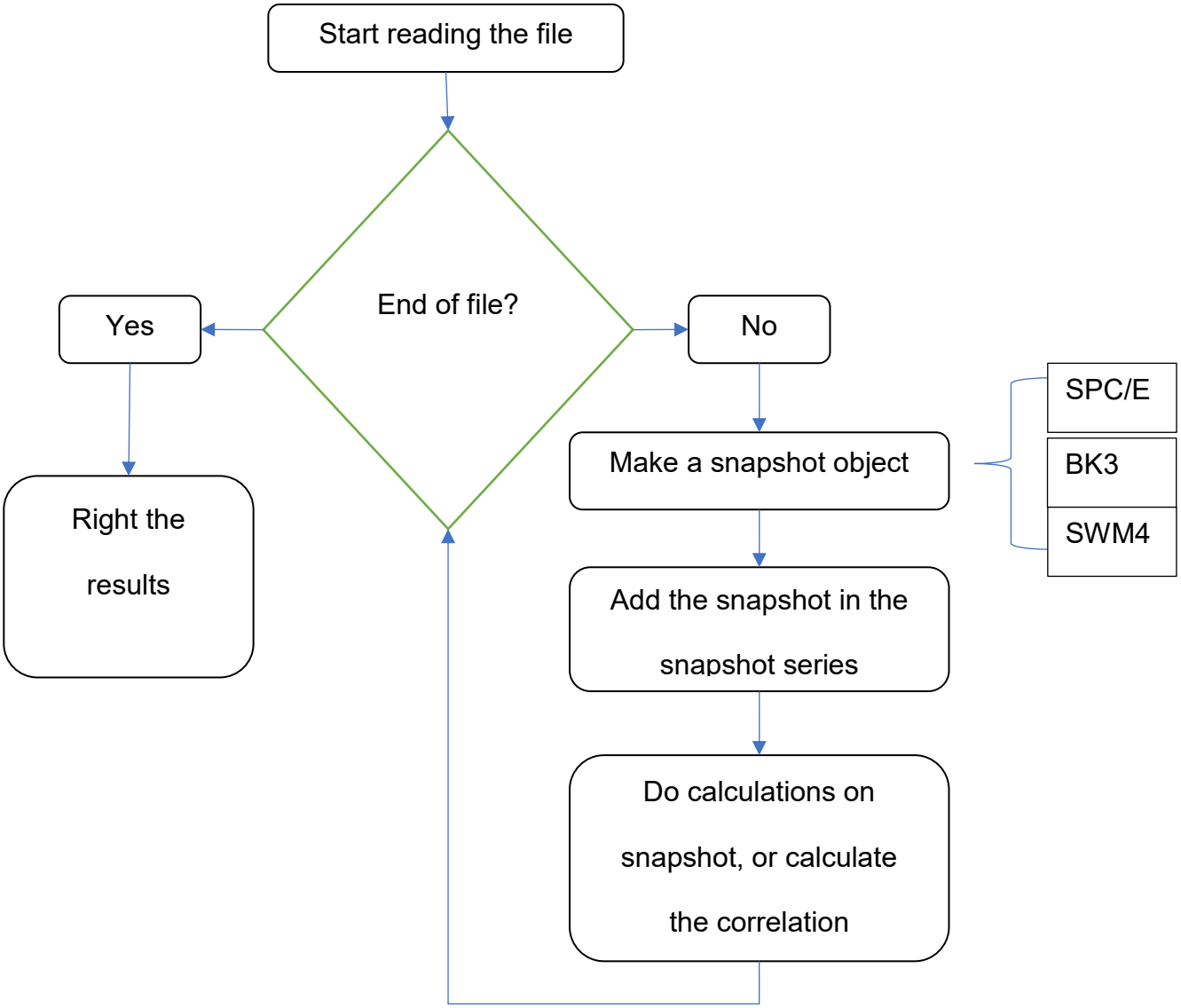
Loop On the E-fields

The calculation folder

```
     #***** step 2 running the skipper

mkfifo  $folderMSDMSR/pipe_skipper

awk 'BEGIN{i=0;p=0;}\        Numerical calculations are done using awk
{i++;\
  if(int(i/1545)==(i/1545)){j++; p=0;\
    if(int(j/10)==(j/10)){p=1;}\
  }\
 if(p==1){print $0;}\
}' <  $folderEfield/nopipe_one > $folderMSDMSR/pipe_skipper  &




cd $folderMSDMSR


./corrCoss_9 pipe_skipper setup_file &  Running the post-processing code

echo "E field $i runned " >>  $folderf/runned_MSD  Reporting the value to the main folder


done
# end loop Efield


done
#end loop freq
```

5. Post-processing programs

Despite the differences in the models, the calculation part for each quantity is the same for all models. For example, the code for calculating the mean square displacement is the same for three models, but the difference is in how the trajectories are read into the program. Here is a simple chart of all the post processing programs:



5.1. Snapshot reading

The following codes read one full snapshot from the trajectory and make the snapshot objects:

For SPC/E:

```
while(  fscanf(dumpfile, "%s\n", check) != EOF ){
   // first we have to find an identifier
```

Read the whole file

```
   if(strcmp(check,identifier)==0){
```
Each trajectory style has an identifier. By reading this we realize that a new snapshot has began

```
        // then there are a some useless words between identifier and the start of
the data lines
      for(int l=1; l<=useless_words; l++){

       fscanf(dumpfile, "%s", &check);

       if(l==1){double sim_time=atof(check);
```
Reading one lone a file, taking care of the boundary conditions
```
time_phase=int((fmod(sim_time,period_T)/period_T)/time_bin); }
       if(strcmp(check,"BOX")==0){
              fscanf(dumpfile, "%s%s%s%s\n", &check2,&check2,&check2,&check2);
               fscanf(dumpfile, "%lf%lf%lf%lf%lf%lf\n",
                    &lXmin,&lXmax,&lYmin,&lYmax,&lZmin, &lZmax);
              lx=lXmax-lXmin; ly=lYmax-lYmin; lz=lZmax-lZmin;

         }

      }

      // now we are in a new time step and we have #Nofatom position. but it may be
interrupted at the middle because
      // of closing lammps. so we have to use while loop. the criteria for end of
file is repeating atom ID

      int iterator=0, nonsense=0, old_atom_id=-1;

      while (iterator< NofAtoms && nonsense<NofAtoms*2){

        //iterator++;
        fscanf(dumpfile, "%i", &atom_id);
        if(atom_id< NofAtoms && atom_id!=old_atom_id){
          iterator++;
          fscanf(dumpfile, "%i%lf%lf%lf", &tid, &xx, &yy, &zz );  //dump: atom syle


          int molecule_number=int(atom_id/3)+1;
          int atom_in_mol= (int) fmod(atom_id,3);

          if( atom_in_mol==1){
```
In SPCE we have one oxygen, and two hydrogens
```
                  X[molecule_number]=lXmin+xx*lx;  Y[molecule_number]=lYmin+yy*ly;
Z[molecule_number]=lZmin+zz*lz;
               //  if( }
                // else{printf("iterator %i mol number %i\n",iterator,
molecule_number);}
             }

          if ( atom_in_mol==2) {
            X_h1[molecule_number]=lXmin+xx*lx;
            Y_h1[molecule_number]=lYmin+yy*ly;
            Z_h1[molecule_number]=lZmin+zz*lz; }
          if( atom_in_mol==0) {
            X_h2[molecule_number-1]=lXmin+xx*lx;
            Y_h2[molecule_number-1]=lYmin+yy*ly;
```

```
            Z_h2[molecule_number-1]=lZmin+zz*lz; }

         old_atom_id=atom_id;
       }
       else{nonsense++;}
```
There can be useless words in the header of the snapshot

```
      }
```

```
      if(iterator==NofAtoms){
```
Do the calculations, only if the snapshot is complete

```
}
```

For BK3:

```
// first we have to find an identifier
   if(strcmp(check,identifier)==0){
```
We start just like SPC/E

```
        // then there are a some useless words between identifier and the start of
the data lines
      for(int l=1; l<=useless_words; l++){

       fscanf(dumpfile, "%s", &check);
       if(strcmp(check,"CRYST1")==0){
                fscanf(dumpfile, "%lf%lf%lf\n",  &lXmax,&lYmax,&lZmax);
              lXmin=0; lYmin=0; lZmin=0;
              lx=lXmax-lXmin; ly=lYmax-lYmin; lz=lZmax-lZmin;
         }

      }

      // now we are in a new time step and we have #Nofatom position. but it may be
interrupted at the middle because
      // of closing lammps. so we have to use while loop. the criteria for end of
file is repeating atom ID

      int iterator=0, nonsense=0;

      while (iterator< NofAtoms && nonsense<20*NofAtoms){


          fscanf(dumpfile, "%s", &check3);
          if(strcmp(check3,"ATOM")==0){

            int molecule_number;

            fscanf(dumpfile, "%i", &atom_id);
            fscanf(dumpfile, "%s", &type);
            fscanf(dumpfile, "%s", &check2);  //useless
            fscanf(dumpfile, "%i", &molecule_number);
            fscanf(dumpfile, "%lf%lf%lf", &xx, &yy, &zz );
```

```
            if (strcmp(type, "OW1") == 0){
```
The central atom is the oxygen atom
```
              if(scaled){
               X[molecule_number]=lXmin+xx*lx;  Y[molecule_number]=lYmin+yy*ly;
Z[molecule_number]=lZmin+zz*lz;}
              else{
               X[molecule_number]=xx;  Y[molecule_number]=yy; Z[molecule_number]=zz;
              }
            }
            if (strcmp(type, "HW2") == 0){
```
The hydrogen atoms are labels as “HW2” or “HW3”
```
              if(scaled){
                X_h1[molecule_number]=lXmin+xx*lx;
Y_h1[molecule_number]=lYmin+yy*ly; Z_h1[molecule_number]=lZmin+zz*lz;
              }
              else{
                X_h1[molecule_number]=xx;  Y_h1[molecule_number]=yy;
Z_h1[molecule_number]=zz;
             }

            }
            if (strcmp(type, "HW3") == 0){
                    if(scaled){
                  X_h2[molecule_number]=lXmin+xx*lx;
Y_h2[molecule_number]=lYmin+yy*ly; Z_h2[molecule_number]=lZmin+zz*lz;
               }
               else{
                  X_h2[molecule_number]=xx;  Y_h2[molecule_number]=yy;
Z_h2[molecule_number]=zz;
               }

            }
            if (strcmp(type, "GM") == 0){
```
The GM particle position is diminished
```
                    if(scaled){
                  X_GM[molecule_number]=lXmin+xx*lx;
Y_GM[molecule_number]=lYmin+yy*ly; Z_GM[molecule_number]=lZmin+zz*lz;
               }
               else{
                  X_GM[molecule_number]=xx;  Y_GM[molecule_number]=yy;
Z_GM[molecule_number]=zz;
               }

            }
            if (strcmp(type, "GH1") == 0){
                    if(scaled){
                  X_GH1[molecule_number]=lXmin+xx*lx;
Y_GH1[molecule_number]=lYmin+yy*ly; Z_GH1[molecule_number]=lZmin+zz*lz;
               }
               else{
                  X_GH1[molecule_number]=xx;  Y_GH1[molecule_number]=yy;
Z_GH1[molecule_number]=zz;
               }

            }
```

```
            if (strcmp(type, "GH2") == 0){
                    if(scaled){
                  X_GH2[molecule_number]=lXmin+xx*lx;
Y_GH2[molecule_number]=lYmin+yy*ly; Z_GH2[molecule_number]=lZmin+zz*lz;
                }
                else{
                   X_GH2[molecule_number]=xx;  Y_GH2[molecule_number]=yy;
Z_GH2[molecule_number]=zz;
                }

            }
            if (strcmp(type, "MW") == 0){
                    if(scaled){
                  X_MW[molecule_number]=lXmin+xx*lx;
Y_MW[molecule_number]=lYmin+yy*ly; Z_MW[molecule_number]=lZmin+zz*lz;
                }
                else{
                   X_MW[molecule_number]=xx;  Y_MW[molecule_number]=yy;
Z_MW[molecule_number]=zz;
                }

            }
```

And for SWM4-NDP:

```
if(strcmp(check,identifier)==0){
```

In lammps the atoms don't have labels. We must distinguish them from their type and their id.

```
                    // then there are a some useless words between identifier and
the start of the data lines
             for(int l=1; l<=useless_words; l++){

                    fscanf(dumpfile, "%s", &check);
                    if(strcmp(check,"BOX")==0){
                           fscanf(dumpfile, "%s%s%s%s\n",
&check2,&check2,&check2,&check2);
                           fscanf(dumpfile, "%lf%lf%lf%lf%lf%lf\n",
                                  &lXmin,&lXmax,&lYmin,&lYmax,&lZmin, &lZmax);
                                  lx=lXmax-lXmin; ly=lYmax-lYmin; lz=lZmax-lZmin;

                    }
             }


              int iterator=0, nonsense=0, old_atom_id=-1;

             while (iterator< NofAtoms && nonsense<NofAtoms*2){

                    //iterator++;
                    fscanf(dumpfile, "%i", &atom_id);
             if(atom_id <= NofAtoms && atom_id!=old_atom_id){

                    iterator++;

                    int tid;
                    double  xx,yy,zz, vx, vy,vz;
```

```
                    fscanf(dumpfile, "%i%lf%lf%lf", &tid, &xx, &yy, &zz );  //dump:
atom syle
                    //printf("%i  %lf    %lf    %lf\n", tid, xx, yy, zz );


                    if(tid==1){

                           int molecule_number=int(atom_id/5)+1;

                           X_ODW_DC[molecule_number]=
unbox(X_ODW_DC[molecule_number], lXmin+xx*lx, lx);
                           Y_ODW_DC[molecule_number]=
unbox(Y_ODW_DC[molecule_number], lYmin+yy*ly, ly);
                           Z_ODW_DC[molecule_number]=
unbox(Z_ODW_DC[molecule_number], lZmin+zz*lz, lz);

                           if(Z_ODW_DC[molecule_number]==0.000){printf("%i   %lf
%lf \n",molecule_number, zz, lz);}
                    }


                    if(tid==3){

                           int molecule_number=int(atom_id/5)+1;

                           X_M[molecule_number]=xx*lx;
                           Y_M[molecule_number]=yy*ly;
                           Z_M[molecule_number]=zz*lz;
                    }

                    if(tid==4){

                           // this is the only one that is filled after
4*NofOxygens. For instance:
                           //   2479     479    4  -1.7162  2.936032e+00 -9.841794e-
01 -1.613161e-01 # ODw SWM4-NDP DP
                           // the molecule id=(atom id)-(4*N of molecule) :
479=2479-4*500
                           int molecule_number=int(atom_id/5);

                           X_ODW_DP[molecule_number]=xx*lx;
                           Y_ODW_DP[molecule_number]=yy*ly;
                           Z_ODW_DP[molecule_number]=zz*lz;
                    }

                    if(tid==2){ we need to label each hydrogen

                           int molecule_number=int(atom_id/5)+1;
                           bool is_h1=true;
                           if(fmod(atom_id,5)==3){is_h1=false;}

             //            int atom_in_mol= (int) fmod(atom_id,4);

                           if (is_h1) {
                                  //X_h1[molecule_number]=xx*lx;
                                  //Y_h1[molecule_number]=yy*ly;
                                  //Z_h1[molecule_number]=zz*lz;

                                  X_h1[molecule_number]= unbox(X_h1[molecule_number],
lXmin+xx*lx, lx);
                                  Y_h1[molecule_number]= unbox(Y_h1[molecule_number],
lYmin+yy*ly, ly);
```

```
                                    Z_h1[molecule_number]= unbox(Z_h1[molecule_number],
lZmin+zz*lz, lz);

                              }
                              else{
                                    //X_h2[molecule_number]=xx*lx;
                                    //Y_h2[molecule_number]=yy*ly;
                                    //Z_h2[molecule_number]=zz*lz;

                                    X_h2[molecule_number]= unbox(X_h2[molecule_number],
lXmin+xx*lx, lx);
                                    Y_h2[molecule_number]= unbox(Y_h2[molecule_number],
lYmin+yy*ly, ly);
                                    Z_h2[molecule_number]= unbox(Z_h2[molecule_number],
lZmin+zz*lz, lz);
                              }
                        }

                        old_atom_id=atom_id;

                }


                        else{nonsense++;}
                }
```

5.2. Calculations

After we made the snapshot we can calculate the properties.

Radial distribution function

```
     //******************************** g(r) and OOO functions
**********************

           //printf("g_z_1=%i \n", g_z_array[1]);

            for(int i=1; i<NofOxygens; i++){
                  for(int j=i+1; j<=NofOxygens; j++){


                  double delta_x=fabs(box(X[i]-X[j],lx)),
```

To make sure that we take care of periodic boundary condition

```
                        delta_y=fabs(box(Y[i]-Y[j],ly)),
                        delta_z=fabs(box(Z[i]-Z[j],lz)),
                        d_r=sqrt(delta_x*delta_x+delta_y*delta_y+delta_z*delta_z);
                     int bin_r=int(floor(d_r/bin_width));

                  if(bin_r> 0 &&  bin_r< g_r_array.size()){
                      g_r_array[floor(d_r/bin_width)]++;
                  }


                  if(delta_x<half_g_square_size && delta_y<half_g_square_size){

                      int binZ=floor(delta_z/bin_width);
```

```
                    if(binZ>0 && binZ< g_z_array.size()){g_z_array[binZ]++;}

                }
```

Cylindrical distribution function

```
                if(delta_y<half_g_square_size && delta_z<half_g_square_size){

                    int binX=floor(delta_x/bin_width);
                   if(binX>0 && binX< g_x_array.size()){g_x_array[binX]++;}
                }

                if(delta_z<half_g_square_size && delta_x<half_g_square_size){

                    int binY=floor(delta_y/bin_width);
                   if(binY>0 && binY< g_y_array.size()){g_y_array[binY]++;}
                }



               }
          }
          n_gr_averaged++;
```

q, and Q6 tetrahedral order parameter

```
      //***************************Q and Q6 part

             for(int i=1; i<NofOxygens-1; i++){

                //printf("size=%i\n", int(neighbor_list[i].size()));
                double temp_cosjk=0;
              int counter=0;
                 //printf("i=%i size=%i\n", i, int(neighbor_list[i].size()) );

                if(int(neighbor_list[i].size())>4){
                  for(int k=0; k<3; k++){
                     double dx1=box(X[i]-X[neighbor_list[i][k]],lx),
                            dy1=box(Y[i]-Y[neighbor_list[i][k]],ly),
                            dz1=box(Z[i]-Z[neighbor_list[i][k]],lz);

                     for(int l=k+1; l<4; l++){
                          double dx2=box(X[i]-X[neighbor_list[i][l]],lx),
                                 dy2=box(Y[i]-Y[neighbor_list[i][l]],ly),
                                 dz2=box(Z[i]-Z[neighbor_list[i][l]],lz);

                        double
cos1=(dx1*dx2+dy1*dy2+dz1*dz2)/sqrt((dx1*dx1+dy1*dy1+dz1*dz1)*(dx2*dx2+dy2*dy2+dz2*d
z2));

                        temp_cosjk+=(cos1+0.333333)*(cos1+0.333333);
                        counter++;
                     }
                    }
                }

                q+=temp_cosjk;
                q_counter++;
```

```
                if(int(neighbor_list[i].size())>11){

                  dbl1D sin_theta(12,0.0),cos_theta(12,0.0), phi(12,0.0),
                        sin_theta2(12,0.0), sin_theta3(12,0.0), sin_theta4(12,0.0),
sin_theta5(12,0.0),  sin_theta6(12,0.0),
                       cos_theta2(12,0.0), cos_theta3(12,0.0), cos_theta4(12,0.0),
cos_theta5(12,0.0), cos_theta6(12,0.0),
                       cos_phi(12,0.0),cos_phi2(12,0.0), cos_phi3(12,0.0),
cos_phi4(12,0.0), cos_phi5(12,0.0), cos_phi6(12,0.0),
                       sin_phi(12,0.0),sin_phi2(12,0.0), sin_phi3(12,0.0),
sin_phi4(12,0.0), sin_phi5(12,0.0), sin_phi6(12,0.0);


                  for(int j=0; j<=11; j++){

                    sin_theta[j]=sin(neighbor_list_theta[i][j]);
                    cos_theta[j]=cos(neighbor_list_theta[i][j]);
                    phi[j]=neighbor_list_phi[i][j];
                    //printf("i=%i  j=%i   phi=%lf \n",i, j, phi[j-4]);

                    sin_theta2[j]=sin_theta[j]*sin_theta[j];
                    sin_theta3[j]= sin_theta[j]*sin_theta2[j];
                    sin_theta4[j]=sin_theta2[j]*sin_theta2[j];
                    sin_theta5[j]=sin_theta4[j]*sin_theta[j];
                    sin_theta6[j]=sin_theta3[j]*sin_theta3[j];

                    cos_theta2[j]=cos_theta[j]*cos_theta[j];
                    cos_theta3[j]= cos_theta[j]*cos_theta2[j];
                    cos_theta4[j]=cos_theta2[j]*cos_theta2[j];
                    cos_theta5[j]=cos_theta4[j]*cos_theta[j];
                    cos_theta6[j]=cos_theta3[j]*cos_theta3[j];

                    cos_phi[j]=cos(phi[j]);
                    cos_phi2[j]=cos(2*phi[j]);
                    cos_phi3[j]=cos(3*phi[j]);
                    cos_phi4[j]=cos(4*phi[j]);
                    cos_phi5[j]=cos(5*phi[j]);
                    cos_phi6[j]=cos(6*phi[j]);

                    sin_phi[j]=sin(phi[j]);
                    sin_phi2[j]=sin(2*phi[j]);
                    sin_phi3[j]=sin(3*phi[j]);
                    sin_phi4[j]=sin(4*phi[j]);
                    sin_phi5[j]=sin(5*phi[j]);
                    sin_phi6[j]=sin(6*phi[j]);
                   }


                              // ok now the main work for Q6:

                  //Y6 -6
                     double zetta=0, gamma=0, betta=0;
                  for(int j=0; j<12; j++){ gamma+= cos_phi6[j]*sin_theta6[j];
betta+=sin_phi6[j]*sin_theta6[j];

                  }
                  //zetta+=y6_6*y6_6*(gamma*gamma-betta*betta)/72.;
                  zetta+=y6_6*y6_6*(gamma*gamma)/72.;

                  gamma=0; betta=0;

                  //Y6 -5
```

```
                    for(int j=0; j<12;
j++){gamma+=cos_phi5[j]*sin_theta5[j]*cos_theta[j];
                                        betta+=sin_phi5[j]*sin_theta5[j]*cos_theta[j];}
                    //zetta+=y6_5*y6_5*(gamma*gamma-betta*betta)/72.;
                    zetta+=y6_5*y6_5*(gamma*gamma)/72.;

                    gamma=0; betta=0;

                    //Y6 -4
                    for(int j=0; j<12;
j++){gamma+=cos_phi4[j]*sin_theta4[j]*(11*cos_theta2[j]-1);

betta+=sin_phi4[j]*sin_theta4[j]*(11*cos_theta2[j]-1);}

                    //zetta+=y6_4*y6_4*(gamma*gamma-betta*betta)/72.;
                       zetta+=y6_4*y6_4*(gamma*gamma)/72.;
                    gamma=0; betta=0;

                    //Y6 -3
                    for(int j=0; j<12;
j++){gamma+=cos_phi3[j]*sin_theta3[j]*(11*cos_theta3[j]-3*cos_theta[j]);

betta+=sin_phi3[j]*sin_theta3[j]*(11*cos_theta3[j]-3*cos_theta[j]);}
                    //zetta+=y6_3*y6_3*(gamma*gamma-betta*betta)/72.;
                    zetta+=y6_3*y6_3*(gamma*gamma)/72.;
                    gamma=0; betta=0;

                    //Y6 -2
                    for(int j=0; j<12;
j++){gamma+=cos_phi2[j]*sin_theta2[j]*(33*cos_theta4[j]-18*cos_theta2[j]+1);

betta+=sin_phi2[j]*sin_theta2[j]*(33*cos_theta4[j]-18*cos_theta2[j]+1);}
                    //zetta+=y6_2*y6_2*(gamma*gamma-betta*betta)/72.;
                    zetta+=y6_2*y6_2*(gamma*gamma)/72.;

                    gamma=0; betta=0;

                    //Y6 -1
                    for(int j=0; j<12;
j++){gamma+=cos_phi[j]*sin_theta[j]*(33*cos_theta5[j]-
30*cos_theta3[j]+5*cos_theta[j]);

betta+=sin_phi[j]*sin_theta[j]*(33*cos_theta5[j]-30*cos_theta3[j]+5*cos_theta[j]);}
                    //zetta+=y6_1*y6_1*(gamma*gamma-betta*betta)/72.0;
                    zetta+=y6_1*y6_1*(gamma*gamma)/72.0;
                    gamma=0; betta=0;

                    //Y6 0
                    for(int j=0; j<12; j++){ gamma+=(231*cos_theta6[j]-
315*cos_theta4[j]+105*cos_theta2[j]-5); }
                    zetta+=y6_0*y6_0*gamma*gamma/144.0;

                    Q6_counter++;
                    double temp_q6=sqrt(0.966643893*zetta);
                    Q6+=temp_q6;
                    //printf("temp_q6=%lf \n", temp_q6);
                    //if(temp_q6>1.0)
                    int q6_dist_bin=int(temp_q6/q6_bin_width);
                    if(q6_dist_bin>0 && q6_dist_bin<q6_dist.size()) {
q6_dist[q6_dist_bin]++; }
```

```
            }  // end if size of neighbor list >=15

          }   // end of loop for calculating q
```

Oxygen triplet angle

```
   //******************************** OOO ************************

      // Hint1: I assume that the neighbor list has calculated in the same step
that I am calculating the OOO
       // Hint2: The neighbors in the neighbor list is arranged based on their
distance from nearerst to farest.

     for(int i=0; i<(int)neighbor_list.size(); i++){ Central oxygen
        double xi=X[i],yi=Y[i],zi=Z[i];


        for(int j=0; j<(int)neighbor_list[i].size()-1; j++){ First neighbor

            int j_index=neighbor_list[i][j];
           double xj=X[j_index],yj=Y[j_index],zj=Z[j_index],
                 dx1=box(xj-xi,lx),dy1=box(yj-yi,ly), dz1=box(zj-zi,lz),
                 r1=sqrt(dx1*dx1+dy1*dy1+dz1*dz1);

            for(int k=j+1; k<(int)neighbor_list[i].size(); k++){ second
neighbor

                  int k_index=neighbor_list[i][k];
                  double  dx2=box(X[k_index]-xi,lx),dy2=box(Y[k_index]-yi,ly),
dz2=box(Z[k_index]-zi,lz),
                         r2=sqrt(dx2*dx2+dy2*dy2+dz2*dz2);  // the o-o vector of
oxygen of molecule #1 to   molecule#2
                  if(r1< oo_max && r2< oo_max){
                     cos_ooo_aver++;
                    double cos_alfa=(dx1*dx2+dy1*dy2+dz1*dz2)/(r1*r2);
                    g_cos_OOO[int((cos_alfa+1.0)/dg_cos)]++;

                    if(fabs(dz1)<half_g_square_size &&
fabs(dz2)<half_g_square_size){
                      cos_ooo_aver_xy++;
                      double cos_alfa_xy=(dx1*dx2+dy1*dy2+dz1*dz2)/(r1*r2);
                      g_cos_OOO_xy[int((cos_alfa_xy+1.0)/dg_cos)]++;
                   }
                  }

            }
        }
     }
```

BK3 dipole moments:

```
          //***************** dipole moment ***************************
```

```
                double muxi,muyi,muzi,instant_dipole2,av_dipole=0.0,
                       mux=0.0,muy=0.0,muz=0.0,
                       mu2_system=0.0,
                       mu_system2=0.0,
                       orient_X=0.0,orient_Y=0.0,orient_Z=0.0,
temp_cos=0.0;
                int dipole_counter=0;
                for(int i=1; i<NofOxygens-1; i++){

                 double  dx_h1=box(X_h1[i]-X[i],lx),
```

We calculate the position of the Drude particles relative to the central oxygen

```
                        dy_h1=box(Y_h1[i]-Y[i],ly),
                        dz_h1=box(Z_h1[i]-Z[i],lz),

                        dx_h2=box(X_h2[i]-X[i],lx),
                        dy_h2=box(Y_h2[i]-Y[i],ly),
                        dz_h2=box(Z_h2[i]-Z[i],lz),

                        dx_GM=box(X_GM[i]-X[i],lx),
                        dy_GM=box(Y_GM[i]-Y[i],ly),
                        dz_GM=box(Z_GM[i]-Z[i],lz),

                 dx_GH1=box(X_GH1[i]-X[i],lx),
                        dy_GH1=box(Y_GH1[i]-Y[i],ly),
                        dz_GH1=box(Z_GH1[i]-Z[i],lz),

                 dx_GH2=box(X_GH2[i]-X[i],lx),
                        dy_GH2=box(Y_GH2[i]-Y[i],ly),
                        dz_GH2=box(Z_GH2[i]-Z[i],lz),

                 delta_orient_x=dx_h1+dx_h2,
                        delta_orient_y=dy_h1+dy_h2,
                            delta_orient_z=dz_h1+dz_h2;

                  muxi=-1.168*dx_GM+0.584*dx_GH1+0.584*dx_GH2;
```

dipole moment calculation

```
                  muyi=-1.168*dy_GM+0.584*dy_GH1+0.584*dy_GH2;
                  muzi=-1.168*dz_GM+0.584*dz_GH1+0.584*dz_GH2;

                  double mu2=muxi*muxi+muyi*muyi+muzi*muzi;

                  Average_dipole=Average_dipole+sqrt(mu2);
                  Average_dipole2=Average_dipole2+mu2;

                  orient_x_system=orient_x_system+ delta_orient_x;
                  orient_y_system=orient_y_system+ delta_orient_y;
                  orient_z_system=orient_z_system+ delta_orient_z;
```

```
                    mu_x_system=mu_x_system+muxi;
                    mu_y_system=mu_y_system+muyi;
                    mu_z_system=mu_z_system+muzi;


                    double dx_temp=delta_orient_x,
dy_temp=delta_orient_y,dz_temp=delta_orient_z,

cos_theta=delta_orient_z/sqrt(delta_orient_x*delta_orient_x+delta_orient_y*
delta_orient_y+delta_orient_z*delta_orient_z);

                    int bin_cos_orient=int((cos_theta+1.0)/cos_bin);
```

Calculating the distribution

```
if(bin_cos_orient<cos_distro.size()){cos_distro[bin_cos_orient]++;}


                    double
cos_theta_dipole=muzi/sqrt(muxi*muxi+muyi*muyi+muzi*muzi);

                    int bin_cos_dipole=int((cos_theta_dipole+1.0)/cos_bin);

if(bin_cos_dipole<cos_distro.size()){cos_distro_dipole[bin_cos_dipole]++;}

                    double
cos_theta_OH=dz_h1/sqrt(dx_h1*dx_h1+dy_h1*dy_h1+dz_h1*dz_h1);

                    int bin_cos_dipole_OH=int((cos_theta_OH+1.0)/cos_bin);


if(bin_cos_dipole_OH<cos_distro_OH.size()){cos_distro_OH[bin_cos_dipole_OH]
++;}

                    //dipole_counter++;
                    dipole_time_average++;

                  }
```

SWM4-NDP dipole moment

```
     //***************** dipole moment **************************
                 double muxi,muyi,muzi,instant_dipole2,av_dipole=0.0,
                        mux=0.0,muy=0.0,muz=0.0,
                        mu2_system=0.0,
                        mu_system2=0.0,
                        orient_X=0.0,orient_Y=0.0,orient_Z=0.0;
                 int dipole_counter=0;
```

```
                for(int i=1; i<NofOxygens-1; i++){

                  muxi=-0.8476*X[i]+0.4238*X_h1[i]+0.4238*X_h2[i];
                  muyi=-0.8476*Y[i]+0.4238*Y_h1[i]+0.4238*Y_h2[i];
                  muzi=-0.8476*Z[i]+0.4238*Z_h1[i]+0.4238*Z_h2[i];


                  instant_dipole2=muxi*muxi+muyi*muyi+muzi*muzi;
                  av_dipole=av_dipole+sqrt(instant_dipole2);
                  dipole_counter++;

                  mux=mux+muxi;
                  muy=muy+muyi;
                  muz=muz+muzi;

                  mu2_system=mu2_system+instant_dipole2;

                  double dx_h1=box(X_h1[i]-X[i],lx),
                         dy_h1=box(Y_h1[i]-Y[i],ly),
                         dz_h1=box(Z_h1[i]-Z[i],lz),
                         dx_h2=box(X_h2[i]-X[i],lx),
                         dy_h2=box(Y_h2[i]-Y[i],ly),
                         dz_h2=box(Z_h2[i]-Z[i],lz),
                         delta_orient_x=dx_h1+dx_h2,
                         delta_orient_y=dy_h1+dy_h2,
                         delta_orient_z=dz_h1+dz_h2;

                  orient_X+=delta_orient_x;
                  orient_Y+=delta_orient_y;
                  orient_Z+=delta_orient_z;


                  double
bin_orient_z=delta_orient_z/sqrt(delta_orient_x*delta_orient_x+delta_orient
_y*delta_orient_y+delta_orient_z*delta_orient_z),

bin_mu_z=muzi/sqrt(muxi*muxi+muyi*muyi+muzi*muzi),
                           temp_phase=step*dumstep*dt*one_on_T;
                           time_phase=int((temp_phase-
int(temp_phase))/time_bin);
                    if(time_phase< distro_orient_z.size() &&
int((bin_orient_z+1.0)/dg_cos) < distro_orient_z[0].size() )
                       {

        distro_orient_z[time_phase][int((bin_orient_z+1.0)/dg_cos)]++;

        distro_mu_z[time_phase][int((bin_mu_z+1.0)/dg_cos)]++;
                         distro_orient_counter[time_phase]++;
                       }
```

```
                    else{printf(" Somethis is wrong with size %i  %i temp
phase %lf \n", time_phase, int((bin_orient_z+1.0)/dg_cos) ,temp_phase);}

                }

                double mu_syste_x=mux/dipole_counter,
                       mu_syste_y=muy/dipole_counter,
                       mu_syste_z=muz/dipole_counter;

                mu2_system=mu2_system/dipole_counter;


mu_system2=(mu_syste_x*mu_syste_x+mu_syste_y*mu_syste_y+mu_syste_z*mu_syste
_z);

                double orient_X_system=orient_X/dipole_counter,
                       orient_Y_system=orient_Y/dipole_counter,
                       orient_Z_system=orient_Z/dipole_counter,

        cos_theta_Z=orient_Z_system/sqrt(orient_X_system*orient_X_system+or
ient_Y_system*orient_Y_system+orient_Z_system*orient_Z_system);

                Average_dipole+=av_dipole/dipole_counter;
                Average_mu2+=mu2_system;
                Average_mu_system2+=mu_system2;
                Average_cos_theta_Z+=cos_theta_Z;
                dipole_time_average++;
```

Detecting hydrogen bonds

```
for(int i=1; i<=NofOxygens; i++){   // for i on oxygen indeces

               //  vicinity_state[i].clear();
                 //int i_water=floor(i/3)+1;
                 for(int jj=0; jj<(int)neighbor_list[i].size(); jj++){    // jj is
just a counter over neighbors of oxygen i
```

Checking the O-O distance criteria

```
                        int j=neighbor_list[i][jj]; // j_water=floor(j/3)+1;
// j which is the id of a neighbor oxygen, i_water is Molecule number
                    double dx=box(X_ODW_DP[i]-X_ODW_DP[j],lx),dy=box(Y[i]-
Y[j],ly), dz=box(Z[i]-Z[j],lz),r=dx*dx+dy*dy+dz*dz; // r is the square of oo
distance
                    bool bond=false;

                       if(r<(oo_max*oo_max)){

                           vicinity_state[i].push_back(j);
```

```
                        // now we can check the hbond:
```

Calculating O-H distance for all 4 possible ways of bonding

```
                        double dx2=box(X_ODW_DP[i]-X_h1[j],lx),dy2=box(Y[i]-
Y_h1[j],ly), dz2=box(Z[i]-Z_h1[j],lz), r2=dx2*dx2+dy2*dy2+dz2*dz2,  // the o-h
vector of oxygen of molecule #1 to  hydrogen #1 of melecule #2
                        dx3=box(X_ODW_DP[i]-X_h2[j],lx),dy3=box(Y[i]-Y_h2[j],ly),
dz3=box(Z[i]-Z_h2[j],lz), r3=dx3*dx3+dy3*dy3+dz3*dz3,
                        dx4=box(X_ODW_DP[j]-X_h1[i],lx),dy4=box(Y[j]-Y_h1[i],ly),
dz4=box(Z[j]-Z_h1[i],lz), r4=dx4*dx4+dy4*dy4+dz4*dz4,
                        dx5=box(X_ODW_DP[j]-X_h2[i],lx),dy5=box(Y[j]-Y_h2[i],ly),
dz5=box(Z[j]-Z_h2[i],lz), r5=dx5*dx5+dy5*dy5+dz5*dz5,
                        coef1=in_molecule_oh*in_molecule_oh+r,
denom=1/(2*sqrt(r*in_molecule_oh)); // denome is unique in all 4 conditions that
should be checked
```

Checking O-H and angular criteria at the same time

```
      int h_donated ,mol_acceptor,mol_donor;
                        if(r2<sq_max_r_o_h && (coef1-r2)*denom>c_min){
bond=true; h_donated=(j-1)*2+1;  mol_acceptor=i; mol_donor=j; test_nhbs++;} //
hydrogen #1 of oxygen j have bond with oxygen  i
                        else if(r3<sq_max_r_o_h && (coef1-
r3)*denom>c_min){bond=true; h_donated=(j-1)*2+2;  mol_acceptor=i; mol_donor=j;
test_nhbs++;}// hydrogen #2 of oxygen j have bond with oxygen  i
                        else if(r4<sq_max_r_o_h && (coef1-
r4)*denom>c_min){bond=true; h_donated=(i-1)*2+1;  mol_acceptor=j; mol_donor=i;
test_nhbs++;}// hydrogen #1 of oxygen i have bond with oxygen j
                        else if(r5<sq_max_r_o_h && (coef1-
r5)*denom>c_min){bond=true; h_donated=(i-1)*2+2;  mol_acceptor=j; mol_donor=i;
test_nhbs++;} // hydrogen #2 of oxygen i have bond with oxygen j
```

For calculating the HB correlation, we need a list of HBs

```
                        if(bond==true){

                              total_bonds++;
                                   //first checking of bifurcation
                              if(hb_state_0[h_donated]==0){ //means no
bifurcation
```

Each hydrogen is assigned a number as the H-bond state, it is 0 if the atom is not donated, and it equal to the acceptor ID if it is donated.

```
                                  hb_state_0[h_donated]=mol_acceptor;
                              }
                              else if(hb_state_02[h_donated]==0){ //means
bifurcation

                                 // the simplest thing is updating th hb_state
                                  hb_state_02[h_donated]=mol_acceptor;
                              }
                              else{

                               printf("Error! a hydrogen cannot be donated to more
that on H");
                               total_triple_bond++;
                               //exit(1);
                              }
```

```
                            }


                             // number of molecules is equal to number of oxygen
                               else{  // if the bond was not on, so we add both
molecules to the       H(1-h) of eachother.

                                  F_state[i].push_back(j);
                                  F_state[j].push_back(i);
                            }
```

F-state means the vicinity state

```
                     }
                     else{

                            if(r<(oo_max2*oo_max2)){
                          F_state_2ndshell[i].push_back(j);
                          F_state_2ndshell[j].push_back(i);
                        }
                    }


                     }  // end of jj
                }   // end for i
```

Making a time series of variables

For doing any kind of time correlation function, we need to have the values between two time-steps: $time = 0$ and $time = t$. Whenever we read a snapshot, we can make the latest configuration at time *time=t,* but the configuration at $t = 0$ should have ben stored in memory. In the code below, we save the position of the oxygen atoms in a time series: X_histo is the histogram of the positions, X is the new position and the function *insert* adds X into the beginning of the histogram array.

```
       X_histo.insert(X_histo.begin(), X);
       Y_histo.insert(Y_histo.begin(), Y);
       Z_histo.insert(Z_histo.begin(), Z);
```

Later on, we need to delete the oldest snapshots from the histogram so that the length of our array does not exceed more than what we need.

```
    if((int) X_histo.size()> msd_lenghth){
          X_histo.pop_back();
          Y_histo.pop_back();
          Z_histo.pop_back();
       }
```

Luzar correlation functions

```
//***************Luzar c(t) and n(t) *************************

/*
for(int s=0; s<c_s;s++){
      if(update[s]){
               int max_of_block= (int) hb_corr_histo[s].size();
                 for(int p=0; p<max_of_block ; p++){
                           counter_s_p_classic[s][p]++;
                 }
      }

3 }
*/


 for(int i=0; i<(int) V_state_t.size(); i++){
             for(int jj=0; jj<(int)V_state_t[i].size(); jj++){
                int j=V_state_t[i][jj];
                bool bond1=false,
bondty1=false,bondty2=false,bondty3=false,bondty4=false;
                n_H++;
                //c_t_taggH_sum(c_s,Int1D(c_p,0)),
         //c_t_omega_sum(c_s,Int1D(c_p,0)),
c_t_taggH_omega_sum(c_s,Int1D(c_p,0));
                if(hb_corr_histo[0][0][(i-1)*2+1]==j ||
hb_corr_histo[0][0][(i-1)*2+2]==j || hb_corr_histo[0][0][(j-1)*2+1]==i ||
hb_corr_histo[0][0][(j-1)*2+2]==i){bond1=true; n_hbonds++; }

                if(hb_corr_histo[0][0][(i-1)*2+1]==j){bondty1=true;}
                if(hb_corr_histo[0][0][(i-1)*2+2]==j){bondty2=true;}
                if(hb_corr_histo[0][0][(j-1)*2+1]==i){bondty3=true;}
                if(hb_corr_histo[0][0][(j-1)*2+2]==i){bondty4=true;}
                // now we have to search for history fo bond:
                 for(int s=0; s<c_s;s++){
                       if(update[s]){
                                int max_of_block= (int)
hb_corr_histo[s].size();
                                for(int p=0; p<max_of_block ; p++){
```

```
                                  if(hb_corr_histo[s][p][(i-1)*2+1]==j ||
hb_corr_histo[s][p][(i-1)*2+2]==j || hb_corr_histo[s][p][(j-1)*2+1]==i ||
hb_corr_histo[s][p][(j-1)*2+2]==i){
                                          R_t_sum[s][p]++;
                                          h_plus_h[s][p]++;
                                          if(bond1){
                                            c_t_sum[s][p]++;
h_plus_h[s][p]+=2; n2_t_sum[s][p]++;
                                            c_t_E_sum[s][p]++;

c_t_omega_sum[s][p]=c_t_omega_sum[s][p]+cos(freqt*acctul_time[s*c_p+p]);
                                          }
                                          else{n_t_sum[s][p]++;
n_t_E_sum[s][p]++;}
                                          /*
                                          if( (bondty1==true &&
hb_corr_histo[s][p][(i-1)*2+1]==j) ||
                                              (bondty2==true &&
hb_corr_histo[s][p][(i-1)*2+2]==j) ||
                                              (bondty3==true &&
hb_corr_histo[s][p][(j-1)*2+1]==i) ||
                                              (bondty4==true &&
hb_corr_histo[s][p][(j-1)*2+2]==i)
                                          ){
                                            c_t_taggH_sum[s][p]++;

c_t_taggH_omega_sum[s][p]=c_t_taggH_omega_sum[s][p]+cos(freqt*acctul_time[s
*c_p+p]);
                                          }

else{n_t_taggH_correction[s][p]++;}
                                          */
                                      }
                                      else{
                                        if(bond1){h_plus_h[s][p]++;}
                                        else {n2_t_sum[s][p]--;}
                                      }

                                      //************ tagged part
*************
                                       if(hb_corr_histo[s][p][(i-1)*2+1]==j){

if(bondty1==true){c_t_taggH_sum[s][p]++;  n2_t_sum_taggedH[s][p]++;}
                                        else{n_t_sum_taggedH[s][p]++;
                                          if(bond1){
                                             c_dblPrime_sum[s][p]++;
                                             fprintf(test_nt_file,"step %i s
%i p %i i %i, j %i, bondt1 %i bondt2 %i bondt3 %i bondt4 %i bond01 %i
bond02 %i bond03 %i bond04 %i\n",
```

```
                                                                    step,s,
p,i,j,hb_corr_histo[s][p][(i-1)*2+1],hb_corr_histo[s][p][(i-1)*2+2],

hb_corr_histo[s][p][(j-1)*2+1],hb_corr_histo[s][p][(j-1)*2+2],

hb_corr_histo[0][0][(i-1)*2+1],hb_corr_histo[0][0][(i-1)*2+2],

hb_corr_histo[0][0][(j-1)*2+1],hb_corr_histo[0][0][(j-1)*2+2]);

if(bondty2)c_dblPrime_case_1[s][p]++;
                                          else c_dblPrime_case_2[s][p]++;
                                       }
                                    }
                                 }
                                 else{

if(bondty1==false){n2_t_sum_taggedH[s][p]--;}
                                 }

                                    //number 2
                                 if(hb_corr_histo[s][p][(i-1)*2+2]==j){

if(bondty2==true){c_t_taggH_sum[s][p]++;  n2_t_sum_taggedH[s][p]++;}
                                    else{n_t_sum_taggedH[s][p]++;
                                         if(bond1){
                                           c_dblPrime_sum[s][p]++;

if(bondty1)c_dblPrime_case_1[s][p]++;
                                           else
c_dblPrime_case_2[s][p]++;
                                         }
                                    }
                                 }
                                 else{

if(bondty2==false){n2_t_sum_taggedH[s][p]--;}
                                 }


                                    //number 3
                                 if(hb_corr_histo[s][p][(j-1)*2+1]==i){

if(bondty3==true){c_t_taggH_sum[s][p]++;  n2_t_sum_taggedH[s][p]++;}
                                       else{
                                         n_t_sum_taggedH[s][p]++;
                                         if(bond1){
                                           c_dblPrime_sum[s][p]++;

if(bondty4)c_dblPrime_case_1[s][p]++;
```

```
                                                    else
c_dblPrime_case_2[s][p]++;
                                                 }
                                              }
                                    }
                                    else{

if(bondty3==false){n2_t_sum_taggedH[s][p]--;}
                                    }

                                         //number 4
                                    if(hb_corr_histo[s][p][(j-1)*2+2]==i){

if(bondty4==true){c_t_taggH_sum[s][p]++;  n2_t_sum_taggedH[s][p]++;}
                                           else{n_t_sum_taggedH[s][p]++;
                                             if(bond1){
                                               c_dblPrime_sum[s][p]++;

if(bondty3)c_dblPrime_case_1[s][p]++;
                                                    else
c_dblPrime_case_2[s][p]++;
                                             }
                                          }
                                    }
                                    else{

if(bondty4==false){n2_t_sum_taggedH[s][p]--;}
                                    }

                                 }
                            }
                       }
                  }
             }
```

New correlation functions

As we explained in the in chapter 3, we have calculated all cases of hydrogen bonding and switching states. Here are the correlation functions:

Table 1. In this table, $h_t = 1$ if the donated hydrogen, H^*, is donated to the first acceptor, O_a and zero otherwise. $h_{t2} = 1$ if H^* is donated to the second acceptor, O_b, and zero otherwise. $H(t) = 1$ if the O^* and O_a are in the first

coordination shell, and . $H'(t) = 1$ if the O^* and O_b are in the first coordination shell. For our discussion we needed only a few of them and we chose an appropriate name for them that we write in the right column.

Case ID	Correlation function	Name in the thesis
1	$\frac{< h_t(0)h_t(t)\big(1-h_{t2}(t)\big) >}{< h_t >}$	$c_t(t)$
2	$\frac{< h_t(0)\big(1-h_t(t)\big)H(t)\big(1-h_{t2}(t)\big)(1-H'(t)) >}{< h_t >}$	
2-2	$\frac{< h_t(0)\big(1-h_t(t)\big)(1-H(t))\big(1-h_{t2}(t)\big)H'(t) >}{< h_t >}$	
2-3	$\frac{< h_t(0)\big(1-h_t(t)\big)(1-H(t))\big(1-h_{t2}(t)\big)(1-H'(t)) >}{< h_t >}$	
2-0	Case 2+ case 3	$n_t(t)$[1]
3	$\frac{< h_t(0)\big(1-h_t(t)\big)H(t)\big(1-h_{t2}(t)\big)H'(t) >}{< h_t >}$	
4	$\frac{< h_t(0)\big(1-h_t(t)\big)H(t)\big(h_{t2}(t)\big) >}{< h_t >}$	$n_d(t)$
5	$\frac{< h_t(0)\big(1-h_t(t)\big)(1-H(t))\big(h_{t2}(t)\big) >}{< h_t >}$	$n_s(t)$
6	$\frac{< h_t(0)\big(h_t(t)\big)\big(h_{t2}(t)\big) >}{< h_t >}$	H-bond bifurcation

[1] This is the tagged version of $n(t)$.

And here is the code:

```
for(int i_hydrogen=1; i_hydrogen<=NofHydrogens; i_hydrogen++){

    int i_water=int(i_hydrogen/2+0.5);
    bool bond_t=false;
    int mol_acceptor_t;
    if(hb_corr_histo[0][0][i_hydrogen]>0){  // it means that a bond does exit
h(t)>0)

       bond_t=true;
       mol_acceptor_t=hb_corr_histo[0][0][i_hydrogen];
    }
    for(int s=0; s<c_s;s++){

       if(update[s]){

          int max_of_block= min((int) hb_corr_histo[s].size(),c_p);
          for(int p=0; p<max_of_block ; p++){

             // a usefull variabe
             int the_time_0= the_time_t - course_grain_time_founder(c_p, c_m, s, p);

             if(hb_corr_histo[s][p][i_hydrogen]>0){  // it means that a bond does
exit h(0)>0

                // otherwise nothing

                counter_s_p[s][p]++;

                // to be used later,  we need to find the next bond after the bond at
t=0:
                 bool bond_at0_found=false;
                int i_seq_bonds=0, next_mol_acceptor=0;

                while(i_seq_bonds<hb_seq_histo[i_hydrogen].size() &&
bond_at0_found==false)
                {

if(hb_seq_histo[i_hydrogen][i_seq_bonds]==hb_corr_histo[s][p][i_hydrogen]){

                     if(hb_seq_histo_last[i_hydrogen][i_seq_bonds]> the_time_0 &&
hb_seq_histo_fisrt[i_hydrogen][i_seq_bonds]< the_time_0){

                          bond_at0_found=true;
                           // it's rare, but it can happen that a specific bond
breaks, another bond forms, and the previous bond reforms again

if(i_seq_bonds>0)next_mol_acceptor=hb_seq_histo[i_hydrogen][i_seq_bonds-1];
                     }
                      else{
                        //   bond_returning++;
                            total_bond_return_to_previous_acceptor;
                     }


                   }
```

```
                    i_seq_bonds++;
                }// end while


               if(bond_t){


                   //can be case 1,6,4,5,
                   if(mol_acceptor_t==hb_corr_histo[s][p][i_hydrogen]){  // means
acceptor at t and 0 are the same

                       // case 1, or 6, the difference is the bifurcation

                       // so fistly let's do  the bifurcation, the beauty of the new
algorithm is that only one time bifurcation is measured
                       bool bifurcation_found=false, bifur_time_pased=false;
                       int i_seq_bifurcation=0, bifurcation_bond=0;
                       while (i_seq_bifurcation<bifur_seq_histo[i_hydrogen].size() &&
bifurcation_found==false && bifur_time_pased==false)
                       {
                          if(bifur_seq_histo_last[i_hydrogen][i_seq_bifurcation]>
the_time_t){

                             if(bifur_seq_histo_fisrt[i_hydrogen][i_seq_bifurcation]<
the_time_t){

                                bifurcation_found=true;

bifurcation_bond=bifur_seq_histo[i_hydrogen][i_seq_bifurcation];
                             }
                          }
                           else{
                             bifur_time_pased=true;
                          }
                          i_seq_bifurcation++;
                       }
                       // bifurcation is done


                       if(bifurcation_found){
                           // now we have to check if the bifurcation bond is the
next acceptor or not
                           if(next_mol_acceptor==bifurcation_bond){
                              //case 6
                            case6[s][p]++;
                           }
                               else{
                            //  bifurcation_but_not_bond++;
                           }
                     }
                      else{

                        //that is definitely case 1
                        case1[s][p]++;
                     }
                   } // end if the bond of t and 0 are the same

                   else {// so the there is a bond, but this bond is not the same
bond as t=0

                       //firstly let's find is the bond of t=0 still in the first
coordination shell
                       bool still_in=false,still_in_2nsShell=false;
```

```
                        for(int i_F_state=0; i_F_state < F_state_t[i_water].size();
i_F_state++){


if(F_state_t[i_water][i_F_state]==hb_corr_histo[s][p][i_hydrogen])still_in=true;
                       }

                       //printf("ttttthe f2_state_ size= %i \n",
F_state_t_2ndshell[i_water].size());
                        for(int i_F_state_2nsShell=0; i_F_state_2nsShell <
int(F_state_t_2ndshell[i_water].size()); i_F_state_2nsShell++){


if(F_state_t_2ndshell[i_water][i_F_state_2nsShell]==hb_corr_histo[s][p][i_hydrogen])
{still_in_2nsShell=true;}
                         //printf("F state %i  hb corr
%i\n",F_state_t_2ndshell[i_water][i_F_state_2nsShell]);
                         //else{}

                       }

                       //if(still_in_2nsShell){
                        // printf("the f2_state_ size= %i  %i\n",
F_state_t_2ndshell[i_water].size(),check_still_in);

                      //}


                        if(mol_acceptor_t==next_mol_acceptor){ //it can be case 4 or 5

                             if(still_in){
                              //case 4
                              case4[s][p]++;
                            }
                            else{
                              //case 5
                             case5[s][p]++;
                             if(still_in_2nsShell){case5_2[s][p]++;}
                           }
                       }
                       else{  // means that the bond has changed 2 times or more, so
there is still probability that the bond of t=0 is in the first shell
                            bool next_in=false;
                             for(int i_F_state=0; i_F_state <
F_state_t[i_water].size(); i_F_state++){

if(F_state_t[i_water][i_F_state]==next_mol_acceptor)next_in=true;
                             }
                            if(still_in){
                               //case 3N or 2N
                               if(next_in)case3N[s][p]++;
                              else{case2N[s][p]++;}
                            }
                            else{
                              //case 2-2N
                              if(next_in)case2_2N[s][p]++;
                           }

                       }
                    }
                }// bond t
                else{
```

```
                        bool still_in=false;
                      for(int i_F_state=0; i_F_state < F_state_t[i_water].size();
i_F_state++){


if(F_state_t[i_water][i_F_state]==hb_corr_histo[s][p][i_hydrogen])still_in=true;
                      }
                      bool next_in=false;
                      for(int i_F_state=0; i_F_state < F_state_t[i_water].size();
i_F_state++){


if(F_state_t[i_water][i_F_state]==next_mol_acceptor)next_in=true;
                       }
                      if(still_in){

                            case2_0[s][p]++;
                         if(next_in){
                         //case 3
                          case3[s][p]++;
                         // printf(" case 3 s %i p %i \n",s,p);
                        }
                        else{
                         //case 2
                          case2[s][p]++;
                       //    printf(" case 2 s %i p %i \n",s,p);
                        }
                      }
                      else{
                       case2_3[s][p]++;
                       //case 2-2
                        if(next_in){case2_2[s][p]++;}
                      // printf(" case 2-2 s %i p %i \n",s,p);

                      }
                } // else if not bond

             }  //end hb_corr_histo[s][p][i_hydrogen]
          } //end loop on p
        }// end   if(update[s])
    }  // end for(int s=0; s<c_s;s++)
}// end loop on hydrogens
```

Mean square displacement

```
  if((int) direcX_histo.size() != (int) direcY_histo.size()){

     printf(" error! not same size\n");
  }


        if(step>msd_lenghth && fmod(step,msd_step)==0){


            for(int i=2; i<=msd_lenghth; i++){
```

```
                double dx=0.0, dy=0.0, dz=0.0,ds_x,ds_y,ds_z,
d_cos=0.0,cC=0, p2cC=0,p3cC=0,p4cC=0,
                       dphi_temp_x,dphi_temp_y,dphi_temp_z,
                       dphi_q_temp_x,dphi_q_temp_y,dphi_q_temp_z,
                       dphi_r_temp_x,dphi_r_temp_y,dphi_r_temp_z,
                       dph_x=0.0, dph_y=0.0, dph_z=0.0,
                       dph_q_x=0.0, dph_q_y=0.0, dph_q_z=0.0,
                       dph_r_x=0.0, dph_r_y=0.0, dph_r_z=0.0,
                       dphi_dr_v1=0;

                int ensemble_accum=0;


                for(int j=1; j<NofOxygens; j++){

                 //double
valueofdirec_j=sqrt((direcX_histo[i][j]*direcX_histo[i][j]+direcY_histo[i][[
j]*direcY_histo[i][j]+direcZ_histo[i][j]*direcZ_histo[i][j]));
                // double
valueofdirec_0=sqrt((direcX_histo[0][j]*direcX_histo[0][j]+direcY_histo[0][[
j]*direcY_histo[0][j]+direcZ_histo[0][j]*direcZ_histo[0][j]));
                // double
denom_cos=(direcX_histo[i][j]*direcX_histo[0][j]+direcY_histo[i][j]*direcY_
histo[0][j]+direcZ_histo[i][j]*direcZ_histo[0][j]);


//cC+=(direcX_histo[i][j]*direcX_histo[0][j]+direcY_histo[i][j]*direcY_hist
o[0][j]+direcZ_histo[i][j]*direcZ_histo[0][j]);


                   //ds_temp=box(X_histo[i][j]-X_histo[1][j],lx);
                   ds_x=X_histo[i][j]-X_histo[0][j];
                   dx+=ds_x*ds_x;
                   //ds_temp=box(Y_histo[i][j]-Y_histo[1][j],ly);
                   ds_y=Y_histo[i][j]-Y_histo[0][j];
                   dy+=ds_y*ds_y;
                   //ds_temp=box(Z_histo[i][j]-Z_histo[1][j],lz);
                   ds_z=Z_histo[i][j]-Z_histo[0][j];
                   dz+=ds_z*ds_z;

                   d_cos+=cos_histo[i][j]*cos_histo[1][j];

                   dphi_temp_x=phi_histo_x[i][j]-phi_histo_x[0][j];
                   dph_x+=dphi_temp_x*dphi_temp_x;

                   dphi_temp_y=phi_histo_y[i][j]-phi_histo_y[0][j];
                   dph_y+=dphi_temp_y*dphi_temp_y;

                   dphi_temp_z=phi_histo_z[i][j]-phi_histo_z[0][j];
```

```
                    dph_z+=dphi_temp_z*dphi_temp_z;


                    // ********* for q **************
                    dphi_q_temp_x=phi_q_histo_x[i][j]-phi_q_histo_x[0][j];
                    dph_q_x+=dphi_q_temp_x*dphi_q_temp_x;

                    dphi_q_temp_y=phi_q_histo_y[i][j]-phi_q_histo_y[0][j];
                    dph_q_y+=dphi_q_temp_y*dphi_q_temp_y;

                    dphi_q_temp_z=phi_q_histo_z[i][j]-phi_q_histo_z[0][j];
                    dph_q_z+=dphi_q_temp_z*dphi_q_temp_z;

                    // ********* for r **************
                    dphi_r_temp_x=phi_r_histo_x[i][j]-phi_r_histo_x[0][j];
                    dph_r_x=dph_r_x+dphi_r_temp_x*dphi_r_temp_x;

                    dphi_r_temp_y=phi_r_histo_y[i][j]-phi_r_histo_y[0][j];
                    dph_r_y+=dphi_r_temp_y*dphi_r_temp_y;

                    dphi_r_temp_z=phi_r_histo_z[i][j]-phi_r_histo_z[0][j];
                    dph_r_z+=dphi_r_temp_z*dphi_r_temp_z;


                    av_pdx4+=dphi_temp_x*dphi_temp_x;
                    av_pdy4+=dphi_temp_y*dphi_temp_y;
                    av_pdz4+=dphi_temp_z*dphi_temp_z;

                    dphi_dr_v1+= (ds_x*ds_x+ds_y*ds_y+ds_z*ds_z)*

(dphi_temp_x*dphi_temp_x+dphi_temp_y*dphi_temp_y+dphi_temp_z*dphi_temp_z);


                  //  double
valueofdirec_j=sqrt((direcX_histo[i][j]*direcX_histo[i][j]+direcY_histo[i][[
j]*direcY_histo[i][j]+direcZ_histo[i][j]*direcZ_histo[i][j]));
                 //   double
valueofdirec_0=sqrt((direcX_histo[0][j]*direcX_histo[0][j]+direcY_histo[0][[
j]*direcY_histo[0][j]+direcZ_histo[0][j]*direcZ_histo[0][j]));

                    double
cos_dot=(direcX_histo[i][j]*direcX_histo[0][j]+direcY_histo[i][j]*direcY_hi
sto[0][j]+direcZ_histo[i][j]*direcZ_histo[0][j])/

sqrt((direcX_histo[i][j]*direcX_histo[i][j]+direcY_histo[i][j]*direcY_histo
[i][j]+direcZ_histo[i][j]*direcZ_histo[i][j])*(direcX_histo[0][j]*direcX_hi
sto[0][j]+direcY_histo[0][j]*direcY_histo[0][j]+direcZ_histo[0][j]*direcZ_h
isto[0][j]));
```

```
                    cC+=cos_dot;
                    p2cC+=(3*cos_dot*cos_dot-1)/2;
                    p3cC+=(5*cos_dot*cos_dot*cos_dot-3*cos_dot)/2;
                    p4cC+=(35*cos_dot*cos_dot*cos_dot*cos_dot-
30*cos_dot*cos_dot)/8;

                    ensemble_accum++;

                }


                MSD_x[i]+=dx/ensemble_accum;
                MSD_y[i]+=dy/ensemble_accum;
                MSD_z[i]+=dz/ensemble_accum;

                MSD[i]+=(dx+dy+dz)/ensemble_accum;


                av_pdx3+=dph_x;
                av_pdy3+=dph_y;
                av_pdz3+=dph_z;


                av_pdx2+=dph_x/ensemble_accum;
                av_pdy2+=dph_y/ensemble_accum;
                av_pdz2+=dph_z/ensemble_accum;

                if(i==3){
                       av_pdx+=dph_x/ensemble_accum;
                       av_pdy+=dph_y/ensemble_accum;
                       av_pdz+=dph_z/ensemble_accum;

                }


                MSR_x[i]+=dph_x/ensemble_accum;
                MSR_y[i]+=dph_y/ensemble_accum;
                MSR_z[i]+=dph_z/ensemble_accum;

                MSR[i]+=(dph_x+dph_y+dph_z)/ensemble_accum;

                //*****  q    **********
                MSR_q_x[i]+=dph_q_x/ensemble_accum;
                MSR_q_y[i]+=dph_q_y/ensemble_accum;
                MSR_q_z[i]+=dph_q_z/ensemble_accum;

                MSR_q[i]+=(dph_q_x+dph_q_y+dph_q_z)/ensemble_accum;

                //***** r   *************
```

```
                MSR_r_x[i]+=dph_r_x/ensemble_accum;
                MSR_r_y[i]+=dph_r_y/ensemble_accum;
                MSR_r_z[i]+=dph_r_z/ensemble_accum;

                MSR_r[i]+=(dph_r_x+dph_r_y+dph_r_z)/ensemble_accum;

 (dph_x+dph_y+dph_z), i, MSR[i], ensemble_accum);

                MSD_MSR[i]+=dphi_dr_v1/ensemble_accum;


                MS_cos[i]+=d_cos/ensemble_accum;

                Cos_Corr[i]+=cC/ensemble_accum;
                p2_Cos_Corr[i]+=p2cC/ensemble_accum;
                p3_Cos_Corr[i]+=p3cC/ensemble_accum;
                p4_Cos_Corr[i]+=p4cC/ensemble_accum;


                accumulator[i]++;


            }

        }
```

6. Plotting data

We use Gnuplot for drawing the results. All the results are collected in a folder named :Water_E_field_pictures. In this folder, the final results plus the Gnuplot script for making the plots are present. As an example, the following script makes OOO plot for AC field, figure 37 of the thesis:

```
set encoding iso_8859_1
```

this encoding is necessary for plotting roman alphabets

```
set xlabel "cos({/Symbol q}_{OOO})"
set ylabel 'P(cos({/Symbol q}_{OOO}))'
```

```
#set xrange [2:7]
set yrange [0:0.8]

set border lw 3

set key at -0.15,0.4 font 'Helvetica,21' spacing 1.4

set xtics nomirror #0.5,1.5,7
set ytics nomirro 0,0.2

set style line 10 lc rgb "black" dt (4,8,4,8)
```

The style is the same for almost all the plots

```
set style line 11 lc rgb "green" dt (4,8,4,8)
set style line 12 lc rgb "red" dt (4,8,4,8)
set style line 13 lc rgb "orange" dt (4,8,4,8)
set style line 14 lc rgb "blue" dt (4,8,4,8)
set style line 19 lc rgb "#708090" dt (4,8,4,8)

set style line 20 lc rgb "black" lt -1
set style line 21 lc rgb "green" lt -1
set style line 22 lc rgb "red" lt -1
set style line 23 lc rgb "orange" lt -1
set style line 24 lc rgb "blue" lt -1
set style line 29 lc rgb "#708090" lt -1


set term pdf enhanced dashed font 'Helvetica,22'  size 5in,4in
set output 'OOO_AC_bk3.pdf'

set title "BK3"


set multiplot
```

The data has been transferred to this folder names are meaningful.

```
p 'OOO_E0_BK3'       eve 5   u  1:2 w l  ls 24 lw 2   title "No Field",\
  'OOO_F100_E200_BK3'   eve 5   u  1:2 w l  ls 21 lw 2    title "100 GHz"
,\
  'OOO_F100_E100_BK3'   eve 5   u  1:2 w l  ls 11 lw 2    title ""  ,\
  'OOO_F200_E200_BK3'   eve 5   u  1:2 w l  ls 20 lw 2    title "200GHz"
,\
  'OOO_F200_E100_BK3'   eve 5   u  1:2 w l  ls 10 lw 2    title ""  ,\
  'OOO_F500_E200_BK3'   eve 5   u  1:2 w l  ls 22 lw 2    title "500GHz",\
  'OOO_F500_E100_BK3'   eve 5   u  1:2 w l  ls 12 lw 2    title ""

f(x)=1000.5
```

```
f2(x)=0.0

set key at 8.3,0.405  font "Helvetica,22" spacing 1.4

unset xtics
unset ytics
unset xlabel
unset ylabel
unset border

unset arrow

unset title
```

The following lines are just to show the second legend.

```
p  f(x) w l ls 19  lw 2  title "E=0.1 V/\305" ,\
   f(x) w l ls 29  lw 2  title "E=0.2 V/\305"
```

The above codes are just examples of tens of thousands of lines of code that have been developed for the projects. Each function has several versions, but only the final and the best ones are used in the calculation folder.