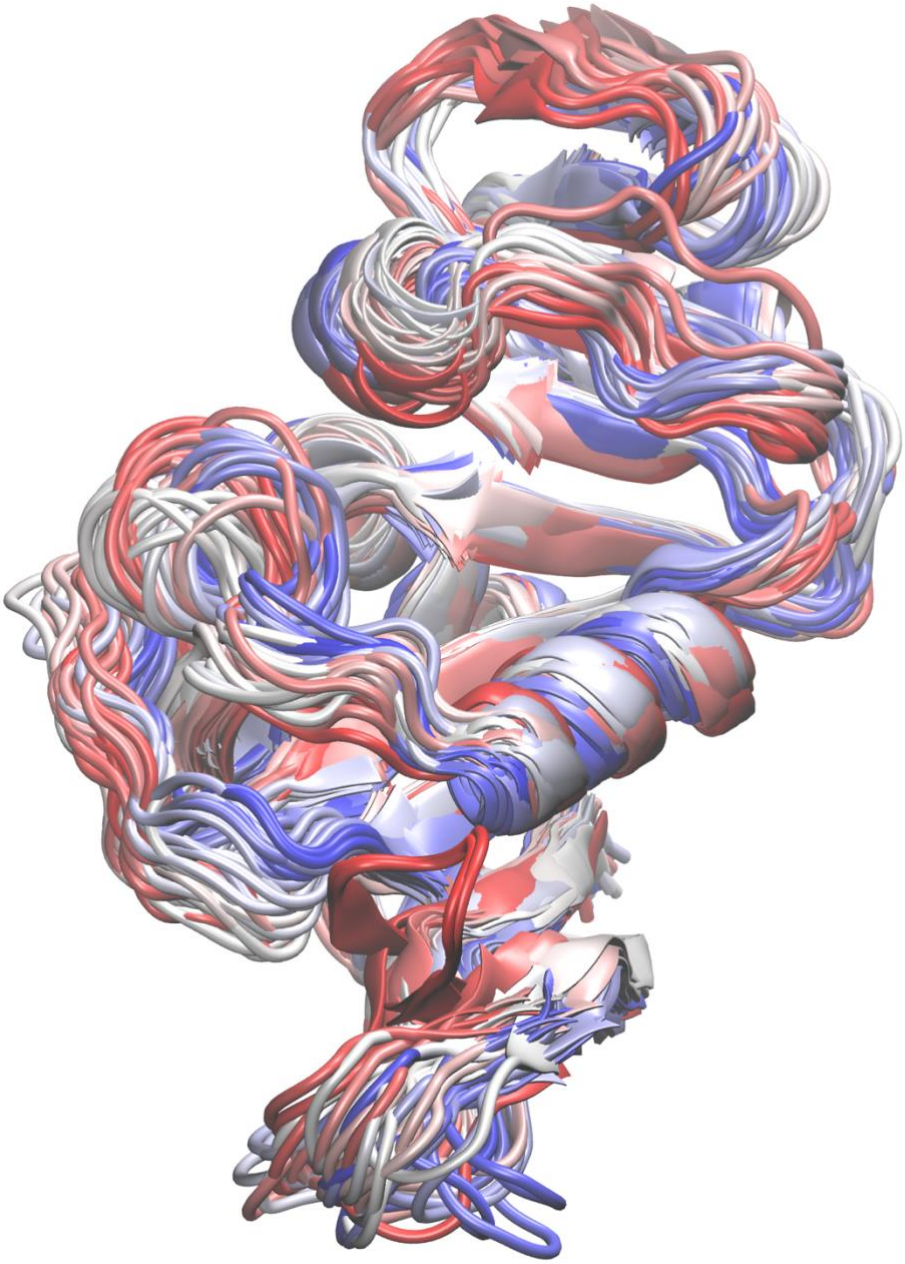
Methyl Side Chain Relaxations and Associated Curves Fit Guide



MIDSTLAB, 2022
Ebru Cetin (ebrucetin@sabanciuniv.edu)

1. Preparation of DCD files

Chopping up the trajectory:

a. raw trajectory is taken, namely traj.dcd
b. chop dcd using catdcd
    i. **catdcd** -o new.dcd -first first time-step -last last-time-step traj.dcd
c. every chopped trajectory should be placed in a different folder

Since side-chain relaxations are on the order of 1-10 ns,[1] we choose the chopping of the trajectory into 10 ns pieces. We have also omitted the first 50ns of the trajectory since our trajectories are reaching the plateau values of RMSD approximately in the first 50 ns. For details on how we arrived at this window size, please see the JCIM 2022 manuscript. This value may be changed according to your bond relaxation behavior or system dynamics.

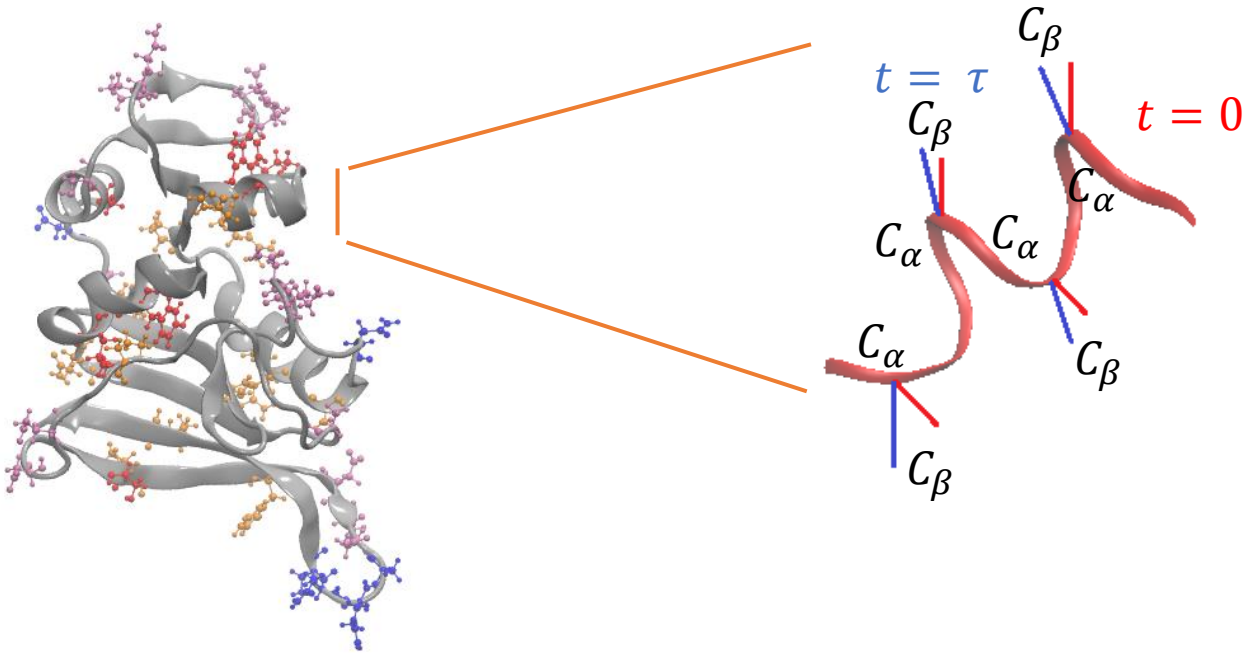


Figure 1. Representation of time-dependent behavior of sidechains; red and blue represent the bond vector at two different time points.

2. Calculation of correlation function

a. **order_param_main.m** should be placed in the first directory and directory names, trajectory name, and corresponding pdb file name should be changed accordingly in this script.

order_param_main:

```
%n is gonna be index vector
1. n=pdbIndex('dhf-a107f.pdb');

[a,b]=size(n);

2. trj = readdcd('dhf-a107f-50-60ns.dcd');

3. ref = trj(1,:);
   [rmsd,trj]=superimpose(ref,trj);

order_p=zeros(159,1);
for residue=1:159
4.     order_p(residue)=corr_func(n(residue,:),residue,trj,10);
end

save(sprintf('Methyl_Order_Parameters_WT.dat'),'order_p','-ascii')
save(sprintf('rmsd.dat'),'rmsd','-ascii')

cd ../60-70ns %2

trj = readdcd('dhf-a107f-60-70ns.dcd');
ref = trj(1,:);
[rmsd,trj]=superimpose(ref,trj);

order_p=zeros(159,1);

for residue=1:159
    order_p(residue)=corr_func(n(residue,:),residue,trj,10);
end

save(sprintf('Methyl_Order_Parameters_WT.dat'),'order_p','-ascii')
save(sprintf('rmsd.dat'),'rmsd','-ascii')
```

1. takes the namd indexing and converts it to mdtoolbox indexes
2. reads the dcd trajectory
3. superimposes the structure on the initial frame
4. calling the correlation function for each of the residue

repeating 2., 3., and 4. step for every chopped time.

b. **pdbIndex.m** and **corr_func.m** functions should be placed in a different folder (such as. Scripts) and this folder should be added to the MATLAB path for future use**.**

pdbIndex.m

```
function [ind] = pdbIndex(filename)
%This function gets a pdb as an input and gives the indexes of atoms
%related with side-chain order parameters
```

```
structure=pdbread(filename);

a=size(structure.Model(1).Atom);
a=a(:,2);
ind=zeros(159,4);
j=1;

for i=1:a
    if strcmp(structure.Model.Atom(i).resName,'MET')==1
        if strcmp(structure.Model.Atom(i).AtomName,'CA')==1
            ind(j,1)=i;
        elseif strcmp(structure.Model.Atom(i).AtomName,'CB')==1
            ind(j,2)=i;
        elseif strcmp(structure.Model.Atom(i).AtomName,'CE')==1
            ind(j,3)=i;
            ind(j,4)=2/3;
            j=j+1;
        end
...
end
ind=(ind-1)*3+1;
end
```

Finds Methionine

Finds desired atom and saves its index to ind array

Here, ‘CE’ is not necessary to store. We only need the ‘CA’- ‘CB’ bond for our analysis; CE is included to exemplify how other atoms may be incorporated into *corr_func.m*. Any atom on the side chain can be chosen for relaxation analysis; ‘CE’ is shown as an alternative.

Correlation Functions

Since all the molecules are wiggling at the same time, their fluctuations as a function of time do not provide direct information.

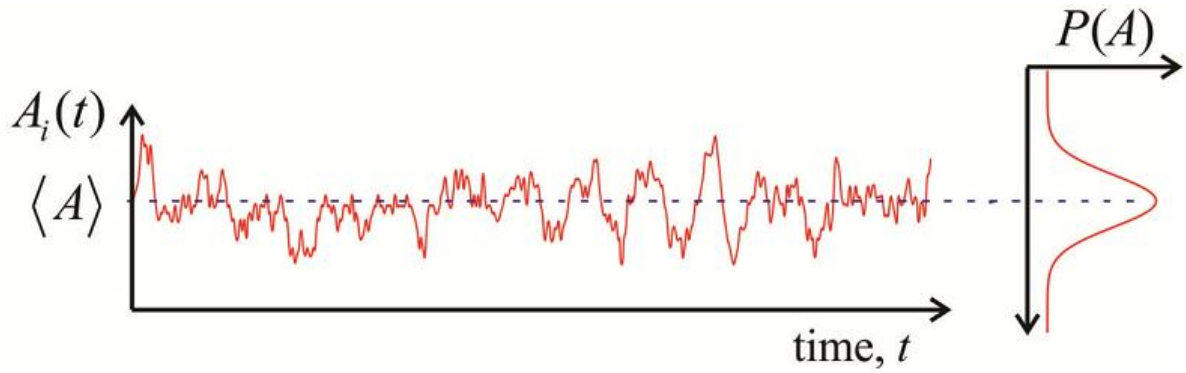


Figure 2. Fluctuations of *A* according to time. The figure is taken from Ref.4.

However, their time-dependent relation with the environment (surroundings) enables us to track down individual time-dependent behavior of a set of molecules. A typical time-correlation function in thermal equilibrium is as follows,

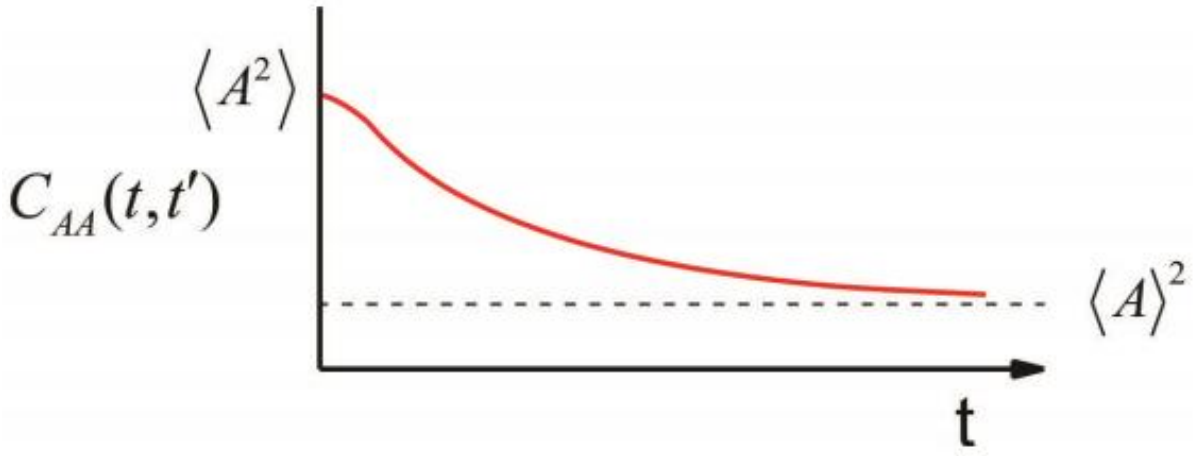


Figure 3. A representative correlation function of time. The figure is taken from Ref.4.

When we take $t = 0$, we reach to the maximum amplitude, mean square values of *A*. When we take $t = \tau$, for long time separations, as thermal fluctuations act to randomize the system, the values of *A* become uncorrelated, hence the value corresponds to square of the mean of *A*. Also, since the system is ergodic, time-dependent data and the ensemble will yield the same result. Conclusively, the limit of correlation function defines the variance for first order correlation functions. However, our correlation function is a second-order Legendre polynomial. Because of that it gives the order parameter. (See Refs. 2,3)

corr_func.m

Inputs and the Output

```
function [ corr_func ] = corr_func( index_list,resid,trj,x )
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here
[m,n]=size(trj);
tao=x*500;
```

This function requires *index_list*, *resid*,*trajectory* and *x*. *index_list* takes the output of pdbIndex or dhfIndex in DHF case. The index file of any desired molecule may be extracted. *resid* takes the residue number, *trajectory* if the dcd file is read in that file, *x* is the length of the trajectory in nanoseconds. Our system is integrated in intervals of 2 fs and the DCD file is collected in every 1000 steps. Thus, we multiply *x* with 500 to find the length of the trajectory. The output of this function is the correlation function or time dependent autocorrelation function.

Our model for the correlation function is,

$$C(t) = S^2 e^{\frac{-t}{\tau_{slow}}} + (1 - S^2) e^{(\frac{-t}{\tau_{fast}})^{\beta}}$$

We have four distinct parameters, S^2 is the order parameter and directly taken out of correlation functions from the limit. The other three parameters are determined by Least Square Optimization of an initial guess.

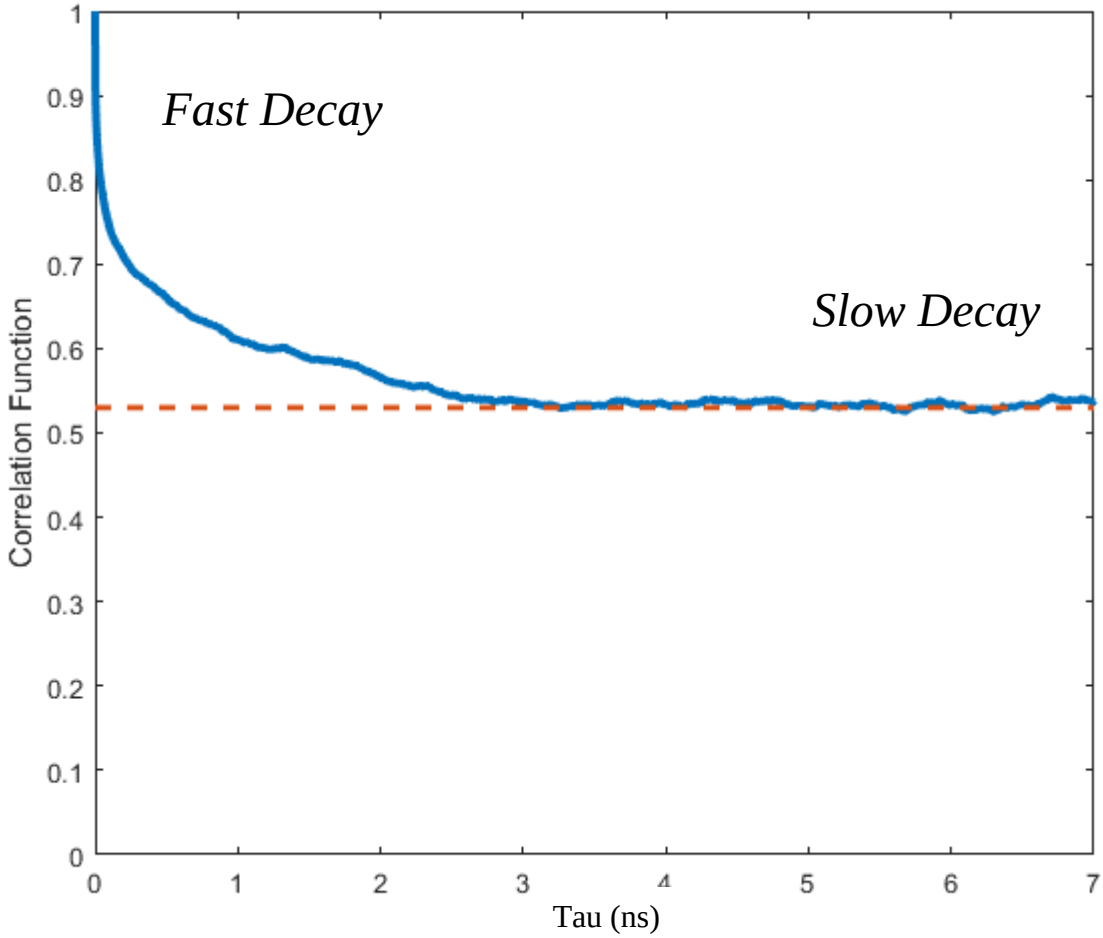


Figure 4. Definition of the correlation function. The long tail is the order parameter. There are two phases of side-chain relaxations, a fast decay and a slow decay.

Initiations and Normalization

```
second_op_axis=zeros(tao,1);
```

Assigning a zero vector to variable that is called second order parameter methyl axis

```
if index_list(1)==0
    corr_func=0;
```

If index list is empty, corr_func is 0.

```
else
    c_a=trj(:,index_list(1):index_list(1)+2); % CA
    c_a=transpose(c_a);

    c_b=trj(:,index_list(2):index_list(2)+2); % CB
    c_b=transpose(c_b);
```

Takes carbon alpha and carbon beta trajectories and resorts them according to time.

Carbon beta-carbon alpha vector calculation

```
%side-chain vector
r_side=c_b-c_a;
```

```
%normalization of the side chain vector
nr_side=zeros(3,m);

for l=1:m
    nr_side(:,l)=r_side(:,l)/norm(r_side(:,l));
end
```

Normalization of the side chain coordinates to get the vectors.

Calculation of the correlation function

```
%%%%%%%% FIRST ORDER CORRELATION FUNCTION %%%%%%%%%%%%%
    g=zeros(tao,tao);
```

```
    for tau=1:tao
        for i=1:m-tau
            g(i,tau)=dot(nr_side(:,i),nr_side(:,i+tau-1));
        end
    end
```

Taking the dot product of CB-CA vector in time *t* and in *t+*

```
    %first order correlation function
    f_corr=zeros(tao,1);
    for tau=1:tao
        f_corr(tau)=sum(g(:,tau))/nnz(g(:,tau));
    end
```

Normalization of first order correlation function with non-zero elements of the array.

The correlation function is calculated as follows: g function calculates the dot product of *t* and $t+\tau$ until τ reaches 10 ns.[2]

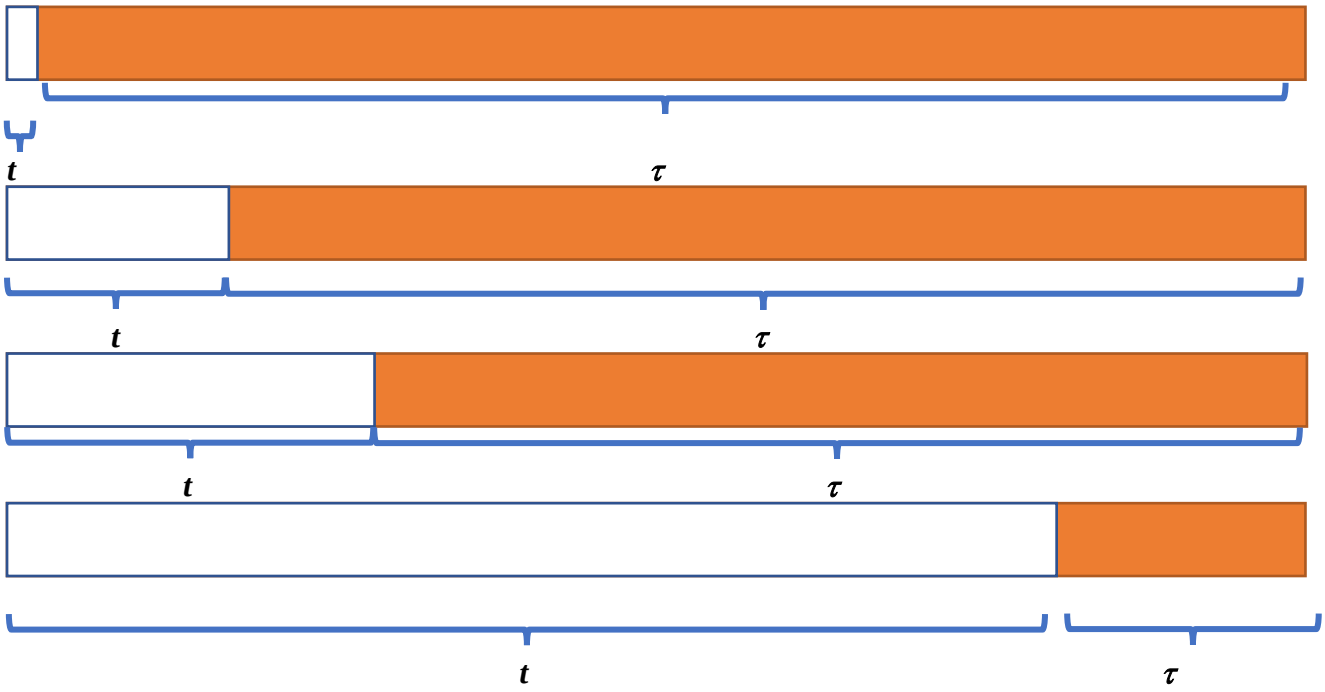


Figure 5. The schematic of the first-order correlation function. Each *t* corresponds to a different τ.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%% SECOND ORDER CORRELATION FUNCTION %%%%%%%%%
```

```
    l_corr=zeros(tao,tao);
```

```
    for i=1:tao
        for j=1:tao
            if g(i,j) ~= 0 % g is left-sided upper triangular
                l_corr(i,j)=1.5*g(i,j).^2-0.5; %operate only on non-zero
                                                % elements
            end
        end
    end
```

Calculation of second-order correlation function. For details, please refer to Ref 2.

```
%nnz divides the number to the count of non-zero elements
% second-order correlation function
    s_corr=zeros(tao,1);
    for i=1:tao
        s_corr(i)=sum(l_corr(:,i))/nnz(l_corr(:,i));
    end
```

Normalization of the second-order correlation function

```
%saving second order correlation function data
    save(sprintf('M_Corr_Func_R%d.dat',resid),'s_corr','-ascii')
    x_corr=linspace(0,x,tao);
    h=figure;
    plot(x_corr,s_corr)
    ylim([0 1])
    title('Methyl Axis Second Order Correlation Function')
    xlabel('Time (ns)')
    ylabel('C(t)')
    savefig(h,sprintf('M_Corr_Func_R%d.fig',resid))
    close(h)
    corr_func=s_corr(end);
```

Saving data file and correlation function figure

c. First folder must have the files.
   i. **order_param_main.m**
   ii. **sample.pdb**
   iii. **sample-chopped.dcd**

the other folders should only contain sample-chopped.dcd's for each time frame. Then **order_param_main** should be executed by changing necessary sample names in the file.

3. Analysis

a. **data_collection.m** should be placed into the first folder (first-time frame) of analysis. The script collects all non-zero positive values recorded for correlation functions. We have 16-time points; so, it collects 16 arrays into a single one. It will create a folder called Average and place the correlation matrix into that folder.
b. **analysis.m**

Calculation of correlation function as an average of 16 chopped pieces

```
corr_average=struct;

s=load('corr_data.mat');
[a,b]=size(s.corr);
[c,d]=size(s.corr(1).Residue);

for i=1:b
    corr_average(i).Func=zeros(5000,1);
    corr_average(i).Std=zeros(5000,1);
    corr_average(i).StdErr=zeros(5000,1);
end



for j=1:c
    for i=1:b
    corr_average(i).Func(j)=mean(s.corr(i).Residue(j,:));
    corr_average(i).Std(j)=std(s.corr(i).Residue(j,:));

corr_average(i).StdErr(j)=std(s.corr(i).Residue(j,:))/sqrt(length(s.corr(i)
.Residue(j,:)));
    end
end
```

order parameter calculation

```
save('corr_average.mat','corr_average')
order_parameter=zeros(1,159);
for i=1:159
    order_parameter(i)=mean(corr_average(i).Func(1500:end));
end

save('order_parameter.mat','order_parameter')
```

Here, we took 3-10 ns tail of correlation function to calculate order parameters.

parameter fits

```
for i=1:159

    s=order_parameter(i);
    ydata=corr_average(i).Func(1:3500);
```

Order parameter and correlation function is fed into the system

Initialization of the fit parameters

```
    if s==0
        xmulti=[0 0 0];
    else
        fitfcn=@(p,xdata)(1-s)*exp(-(xdata/p(1)).^p(2))+(s)*exp(-
xdata/p(3));
```

```
        lb=[0 0 0];
        ub=[Inf 1 Inf];
        p0= [1 0.4 10];
```

Lower, upper and first values of the fit parameters

```
        [xfitted,errorfitted]=lsqcurvefit(fitfcn,p0,xdata,ydata,lb,ub);

        problem=createOptimProblem('lsqcurvefit','x0',
'p0','objective',fitfcn,'lb',lb,'ub',ub,'xdata',xdata,'ydata',ydata);

        ms=MultiStart('PlotFcns',@gsplotbestf);

        [xmulti,errormulti]=run(ms,problem,50);
    end

    tao_fast(i)=xmulti(1);
    tao_slow(i)=xmulti(3);
    beta(i)=xmulti(2);
end
```

Here, MATLAB solves an optimization problem using Least Squares Curve fit to fit a curve to the correlation functions that we have calculated.

saving the parameters

```
end_Parameters.TaoF=tao_fast;
end_Parameters.TaoS=tao_slow;
end_Parameters.Beta=beta;

save('End_Parameters','-struct','end_Parameters')

load('End_Parameters.mat')
load('corr_average.mat')
```

plotting the correlation fit on the correlation function

```
for i=1:159
    ydata=corr_average(i).Func(1:3500);
    xdata=linspace(0,7,3500);
    s=Order(i);

    y=(1-s)*exp(-(xdata/TaoF(i)).^Beta(i))+(s)*exp(-xdata/TaoS(i));
    y=y';

    %calculation of r-square

    Bbar=mean(ydata);
    SStot = sum((ydata-Bbar).^2);
    SSreg = sum((y-Bbar).^2);
```

```
    SSres = sum((ydata-y).^2);
    R2=1-SSres/SStot;

    h=figure;
    plot(xdata,ydata,'b')
    hold on
    plot(xdata,y,'r')
    ylim([0 1])
    xlim([0 7])
    legend('Correlation Function','Fit')
    annotation('textbox', [0.75, 0.7, 0.1, 0.1], 'String', sprintf('R^2:
%2.2f',R2));
    annotation('textbox', [0.3, 0.1, 0.1, 0.1], 'String', sprintf('y=(1-
%1.2f)*exp(-(x/%1.2f)^{%1.2f})+%1.2f*exp(-
x/%4.2f)',s,TaoF(i),Beta(i),s,TaoS(i)));
    title(sprintf('Correlation Function - Residue %d',i))
    savefig(h,sprintf('Corr_and_Fit_R%d.fig',i))
    close(h)
end
```

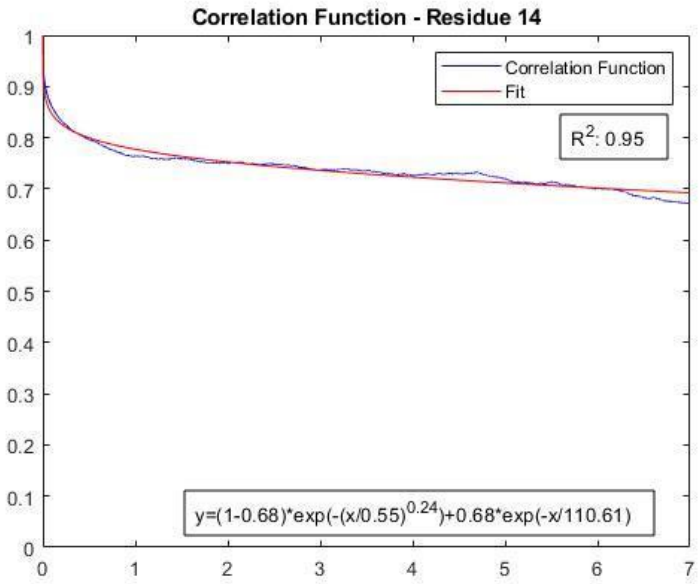


Figure 6. The output of the analyses. A correlation function (blue) and its fit (red). R^2 is at the top right and the equation is shown in the bottom left.

Our model makes this fit using four parameters, S^2, β, τ_{fast}, *and* τ_{slow}. S^2 defines the shape of the well (top right in the figure). β shows the ruggedness inside the well (bottom right in the figure). τ_{fast} is the time spent inside one well (top left). τ_{slow} is the time spent in one overall well (bottom left).

In our analyses, since we are working on 200 ns scaled trajectories, we used $\tau_{slow,}$ as a fitting parameter, and we do not treat it quantitatively. However, the dynamics become clearer its inclusion. Suppose you have one stick attached to a rod and two ropes to rotate it. The general movement will be for small-scale motions unless there is a conformational change. Faster movements of the sidechain will be more dominated by the random forces affecting the system.

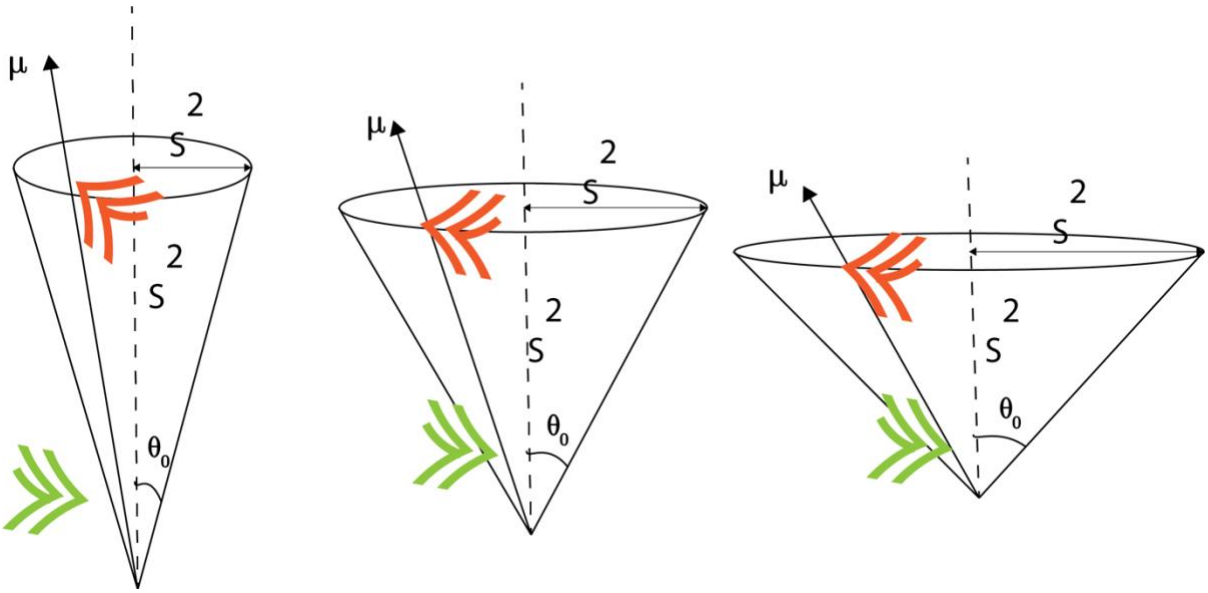


Figure 7. Sharp energy wells to shallower energy wells. S^2 affects both the height and the width of the cone. The green arrow represents slower motions whereas the orange arrows represent random forces in play. For further inquiry on the subject please refer to Ref.3.

These two movements, slow and fast, happen in a restricted environment due to the nature of the sidechain motions. The ruggedness of each energy well is quantified by β. Order parameter (S^2) measures the level of restriction.

References

1. Xu, Y.; Havenith, M., Perspective: Watching low-frequency vibrations of water in biomolecular recognition by THz spectroscopy. *The Journal of Chemical Physics* **2015,** *143* (17), 170901.
2. Best, R. B.; Clarke, J.; Karplus, M., What Contributions to Protein Side-chain Dynamics are Probed by NMR Experiments? A Molecular Dynamics Simulation Analysis. *Journal of Molecular Biology* **2005,** *349* (1), 185-203.
3. G. Lipari and A. Szabo, J. Am. Chem. Soc. **104**, 4559 (1982).
4. "10.1: Definitions, Properties, And Examples Of Correlation Functions - Chemistry Libretexts". *Chem.Libretexts.Org*, **2022**.