prealpha manual

Frederik Philippi

September 30, 2023

A tool to extract information from molecular dynamics trajectories.

Contents

1 Introduction

prealpha is a trajectory analyser which was made for fully atomistic trajectories - mostly cartesian coordinates, however some tools work equally well with trajectories that contain velocities, or even require velocities. Before you dive in here, also have a look at TRAVIS.[3,2] I started coding prealpha after realising that quite a few of the functionalities I needed were not implemented in TRAVIS, thus you will not find absolutely basic things like radial distribution functions in prealpha.* If you decide to use prealpha, please also cite our work.[1,2]

prealpha is written in Fortran and parallelised with the OpenMP library to share the workload. When prealpha (after compilation) is invoked, any command line arguments will be treated as general input files. For every valid one of these input files, the main program runs once. The main program itself consists of two largely independent parts, the user interface and the analysis itself.

The user interface starts only if no general input file could be found. (This can also be one of those specified in the command line) This 'manual' is also available via the user interface. The most useful standalone feature of the user interface is the option to interactively generate input files, which should ensure that the input is formally valid (for example, only lowercase letters are used in the input keywords).

If there is a general input file, then the actual analysis starts, and the general input file is read line by line. During the analyis, no further input is required. Errors in the general input file are not tolerated - if a line cannot be read, then the execution stops and an error is printed. When a keyword linking to a separate input file is found, then the corresponding module is invoked to read and run the separate input. Some of the calculations are quite involved and might run for hours or days. The required real time can be greatly reduced by using parallelisation. To use the parallelisation, it is necessary to load the whole trajectory into RAM. The reason for this is to overcome fileIO as bottleneck.

All output files, including structures, are written to the output folder. These files contain a header with variable names and the reference if necessary. The names of the output files from a certain type of calculation are fixed. This means that if you want to perform, say, two dihedral condition analyses, the files will be overwritten. To avoid this, request a prefix such as 'cation_'.† The 'timeline' printed in some files is obtained from timestep multiplied with the variable TIME_SCALING_FACTOR.‡

*There actually is a way to calculate RDFs with prealpha, but they are only a side product of the polar distribution functions.

†This prefix can be set with 'set_prefix "cation_"'

‡The time scaling factor can be set in the general input file. If you have a trajectory dumped every 1000 steps with 0.5 fs timestep, then a reasonable choice would be 'time_scaling 500', so your time unit will be fs.

2 Main Features

The main features of prealpha are explained here. There are many other small utilities which may be useful, too. These are explained in the section 'input files' as they are invoked straight from the general input file. In contrast to that, each of the following subsections is treated as a distinct feature with its own input file. Some of the more demanding routines exist also in a parallelised version. For most of these features, a number of switches are available. These switches influence e.g. the print level, bin counts or steps to analyse. Information about all possible switches is given in the section 'input files'.

For most of these features, a number of switches are available. These switches influence e.g. the print level, bin counts or steps to analyse. The example input files (option '5' in the main menu) contain the most common switches. Information about all possible switches is provided by option '2' in the main menu.

2.1 Distribution Functions

This feature allows the user to calculate distribution functions of angles and distances. Apart from distribution functions of molecules around each other, this also includes the possibility of calculating the distribution of dipole moments (or charge arms).

2.1.1 Distribution of molecules around each other

Figure 1 shows the cylindrical distribution function of cations around cations in an ionic liquid. There is anisotropy in the system, which can be seen from the higher density perpendicular and collinear to the reference cation. The z-axis was chosen as reference vector here. Normalisation is the same as for a 'classical' RDF, i.e. the distribution function goes to 1 at large distances, corresponding to the ideal gas density.

The same system analysed with a polar distribution function is shown in Figure 2. In Figure 2, the ideal density was subtracted (i.e. the trace, which is exactly the cation-cation-RDF). Thus, the anisotropy becomes even more visible: areas of increased density are positive, those with reduced density are negative.

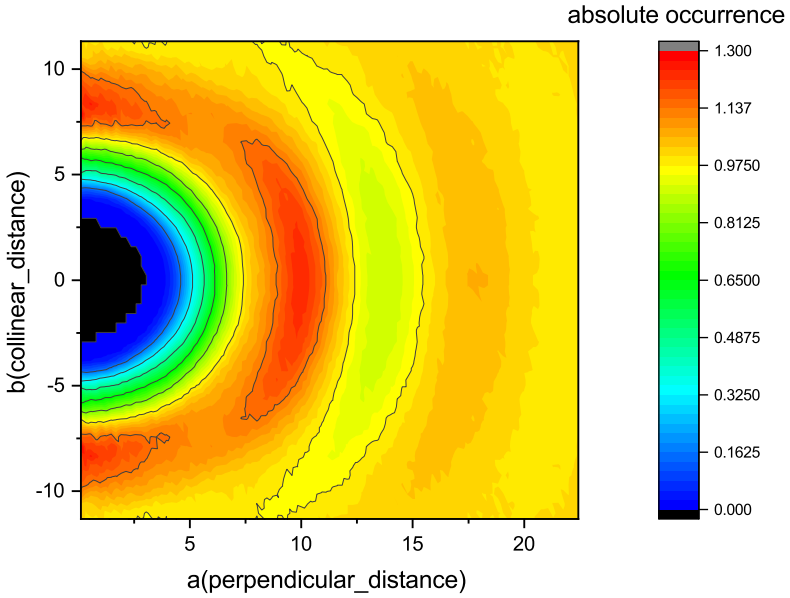

Figure 1: Cylindrical distribution of cations around cations

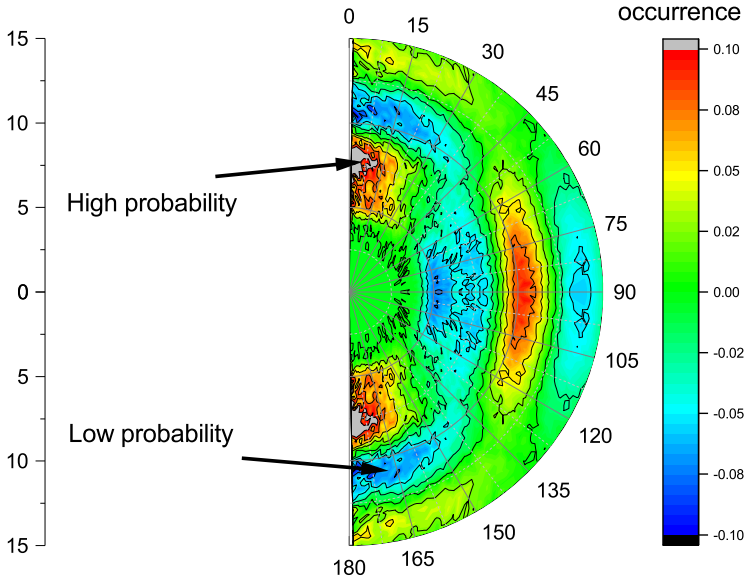

Figure 2: Polar distribution of cations around cations

The distribution functions can also be computed considering *every* molecule type, which corresponds to a normalised sum of all distribution functions. This is useful as the components can then be weighted by charge, hence it is possible to visualise areas of positive and negative charge density in the surroundings of a reference molecule/ion. From this, the Coulomb interaction energy is also accessible as a function of distance, see Figure 3 and Figure 4. This is related to the Sum Rules,[9] which are also implemented in prealpha.

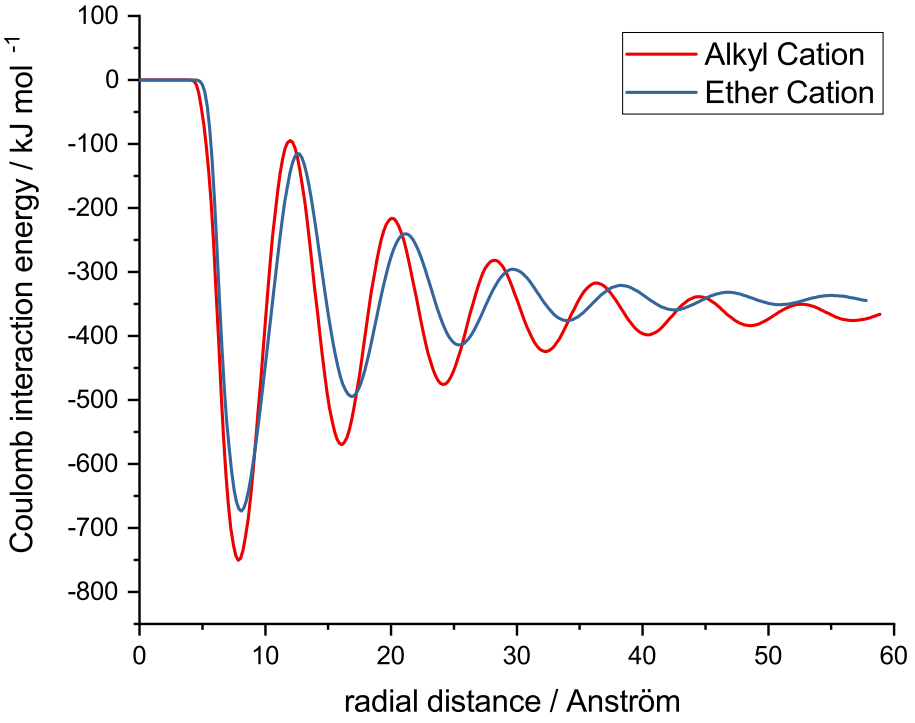

Figure 3: Coulomb interaction energy as a function of distance. Here, the Coulomb stabilisation of a single ion is about 400 kJ/mol.

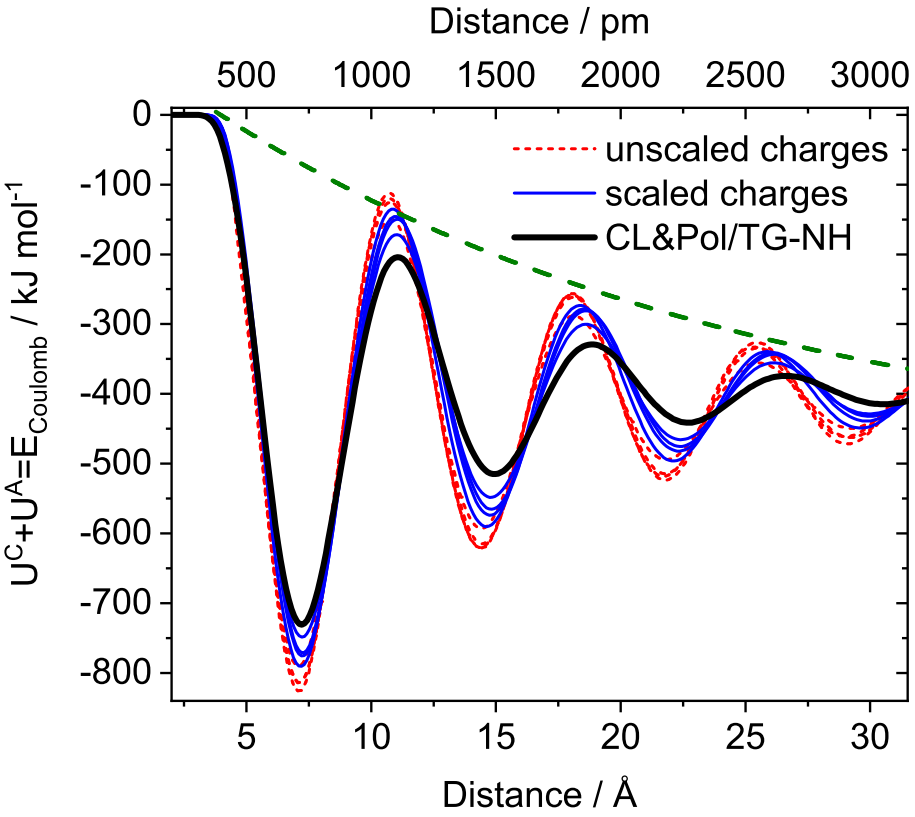

Figure 4: The Coulomb energy integral showing differences between polarisable (CL&Pol/TG-NH), non-polarisable, and scaled charge polarisable simulations. Calculated with integer (± 1) point charges placed on the centre of charge of the ions. The green dashed line is a guide for the eye showing the exponential screening component. Adapted from Ref. [10] with permission from the PCCP Owner Societies.

2.1.2 Charge Arm Distribution

Such a polar distribution function can also be constructed for the dipole moment of the molecules, or the charge arm (distance between center of mass and center of charge) for ions. prealpha also writes the radial distribution function, i.e. the trace of the corresponding polar distribution function. Examples are given in Figure 5 and Figure 6.

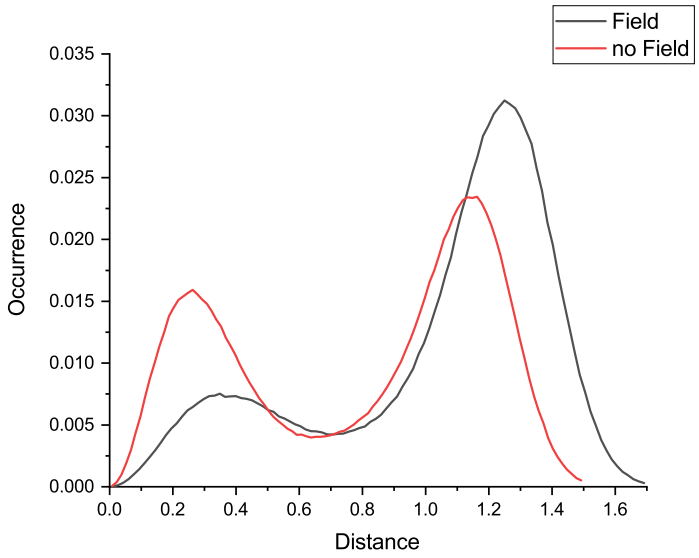

Figure 5: Occurrence of charge arm lengths (distance between centre of charge and centre of mass). The molecule has two states, the one with the larger charge arm is preferred in the presence of an electric field.

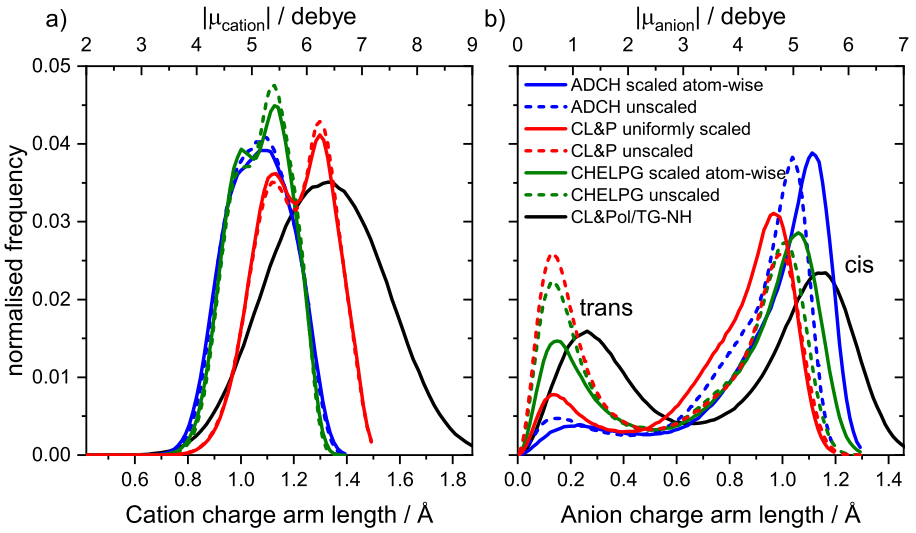

Figure 6: Occurrence of charge arm length in (a) cations and (b) anions of the same ionic liquid, but with different simulation setups. Adapted from Ref. [10] with permission from the PCCP Owner Societies.

2.2 Dihedral Conditions

Allows the user to specify a set of dihedral conditions to be fulfilled. These could e.g. be the two dihedrals in NTf2 ('cisoid' vs. 'transoid'), or some dihedrals along a side chain (= a certain conformation like 'all-trans'). It is possible to 'fold' the specified dihedrals (convenient for cisoid/transoid), then on top of the range 'a' to 'b', prealpha will also check for (360-b) to (360-a). For these conditions, the following analyses are available:

- (Independent) incidences (or 'counts') for each specified dihedral.
- Dependent incidences, i.e. the 2D PES subset (only for 2 dihedrals). From this, the potential of mean force can be obtained as $-RTln(occurrence)$
- For each timestep the share of fulfilled conditions (like, '42.0% transoid')
- The intermittent binary autocorrelation function of the specified condition (PARALLELISED). The definition is given in Equation 1. Here, h is a binary identifier, *i.e.* $h = 1$ if the dihedral condition is fulfilled and $h = 0$ otherwise.

$$C_{dih}(t) = \frac{\langle (h(t_0 + t) - \langle h \rangle)(h(t_0) - \langle h \rangle) \rangle}{\langle (h(t_0) - \langle h \rangle)^2 \rangle} \tag{1}$$

  The lifetime can be obtained from the integral of the autocorrelation function, *i.e.* $\tau_{dih} = \int_0^\infty C_{dih}(t)\mathrm{d}t$.

The encountered values of the specified dihedrals can also be exported in a separate file. The dependent indices are convenient to construct a contour plot showing conformations, Figure 7.

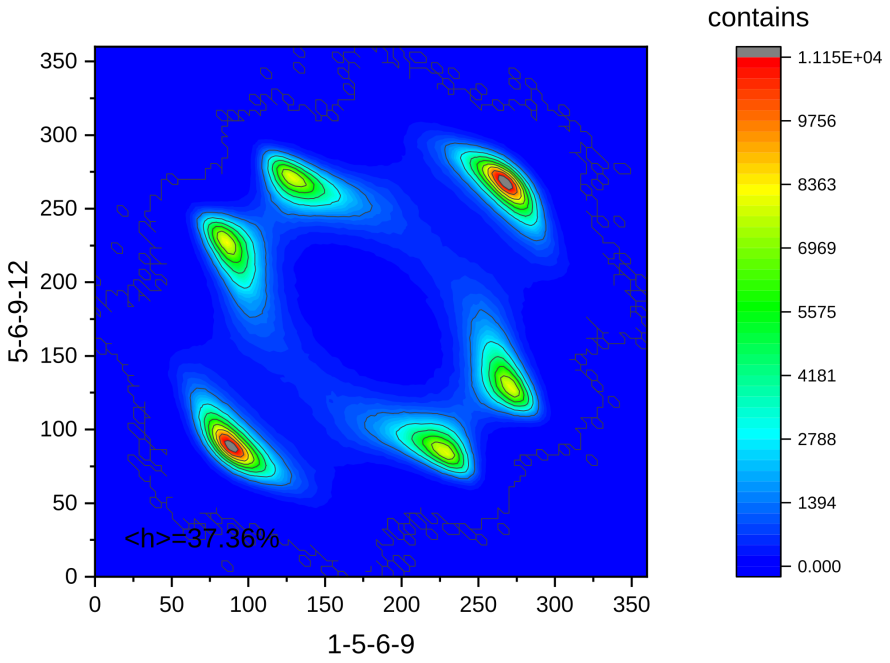

Figure 7: Probability of finding a molecule in a certain geometry (i.e., a certain combination of two dihedral angles). Given as counts.

2.3 Orientation correlation functions

Computes the reorientational time correlation function for a given vector.(PARALLELISED) The base and tip point of this vector are defined as fragment of a molecule (including single atoms) Alternatively, when atomic charges are available, the dipole moment / charge arm vector can be used. Different legendre polynomials are available, and the computed quantity is $C_l(t) = \langle P_l(u(t) \cdot u(t=0)) \rangle$ (u unit vector of fragment, P legendre polynomial of order l, t time shift) see also equation (11.11.1) in[5] or, for ionic liquids, for example equation (2) in[16].

2.4 Relative Mean Molecular Velocity Correlation Coefficients:

Unlike most other modules, this one needs atomic VELOCITIES instead of coordinates. Computes relative mean molecular velocity correlation coefficients based on [15]. No reference frame dependent properties are calculated (who needs these anyway?). Only two molecule types at once are supported currently. If you want cross-correlations, then I would recommend using the Einstein relation rather than the Green-Kubo relation, see also the MSD part. The following quantities from [15] have been implemented:

- RMM-VCFs $\Lambda_{ab}(t)$, equation (4)
- The integral and the normalised function $C_{12}(t)$

Optionally, these self-contributions can also be computed:

- $\Lambda^S(t)$, equation (8), for both specified particles (PARALLELISED)
- The integral and the normalised functions $C_1(t)$ and $C_2(t)$
- Self-velocity correlations, eq (7), as well as $C_0(t)$
- All corresponding diffusion quantities based on eq (17)
- The δ function as in eq (19)
- The time-dependent δ function, calculated as $\delta(t) = C_{12}(t) - C_0(t)$
- A reference frame independent combination of distinct contributions $D^d_{12}(t)$

Thus, everything in Table II. of the above reference is available. Additionally, conductivities are printed:

- Self, distinct and total contributions to the specific electrolytical conductivity
- The same for the molar conductivity (based on total particle number - *2 for ILs)
- Based on that, the predicted Haven Ratio in this framework of theory. Not very accurate unless you have absolutely insanely large trajectories.

The equations for electrolytical conductivity used in this code can be found in [14]. Note that quite a large number of averages has to be taken to obtain sensible values.

2.5 Custom Components of Velocity Correlation Functions

(and electric current autocorrelation function) Unlike most other modules, this one needs atomic VELOCITIES instead of coordinates. You are left to choose the two molecule types to correlate freely. Additionally, you can choose to compute self- or distinct contributions. A good reference is, for example, [7]. Requesting velocity correlation functions will compute the quantities in $\langle ... \rangle$ brackets from equations (A6) to (A10) in [7] - i.e. not including the N, but N_{cat}, N_{an},... Requesting CACFs, however, will compute the quantities in $\langle ... \rangle$ brackets from equation (3). Essentially they are the same, but weighted with charges rather than 1/numbers. Important note: the distinct contributions are extremely expensive to calculate. If possible at all, it is advisable to use the RMM-VCFs, the 'conductivity_simple' keyword, or even the cross contributions from the Einstein relation.

2.6 Mean (Squared) Displacement

Calculates the mean squared displacement including a drift correction.(PARALLELISED) Other exponents can be chosen as well, e.g. for the mean fourth power displacement $\langle R^4 \rangle$. Furthermore, there is the possibility to directly calculate the non-gaussian parameter α_2 and print it together with the mean squared displacement. The diffusion coefficients thus obtained can be used for comparison with the VACFs. Different projections can by chosen by which the displacement vector is to be multiplied. This could be e.g. '1 1 1' (giving the 'standard' 3D diffusion coefficient), or something like '0 0 1' (which would give only the component in z-direction). Two print levels are available. Default is to only print:

- The mean squared displacement $\langle R^2 \rangle$
- The mean displacement $\langle R \rangle$

If the verbose print is requested, then the output additionally contains:

- Drift corrected mean squared displacement $\langle R^2 \rangle - \langle R \rangle^2$
- All three components of the drift vector, $\langle x \rangle$, $\langle y \rangle$, and $\langle z \rangle$
- The number of averages taken to obtain these values.

This module is thus a valuable tool to calculate diffusion when drift is present, Figure 8, see also Reference [4]. A simple example for diffusion without drift is shown in Figure 9.[11] It is also possible to calculate the relative mean molecular diffusion coefficients, using the corresponding Einstein relation.(PARALLELISED) This is much more efficient than the velocity cross correlation coefficients, since the trajectory does not need to be sampled as finely! For my systems, every few picoseconds is fine, whereas the Green-Kubo relation requires sampling every few fs, yet still requiring extremely long runs to converge (on the order of 100 ns).

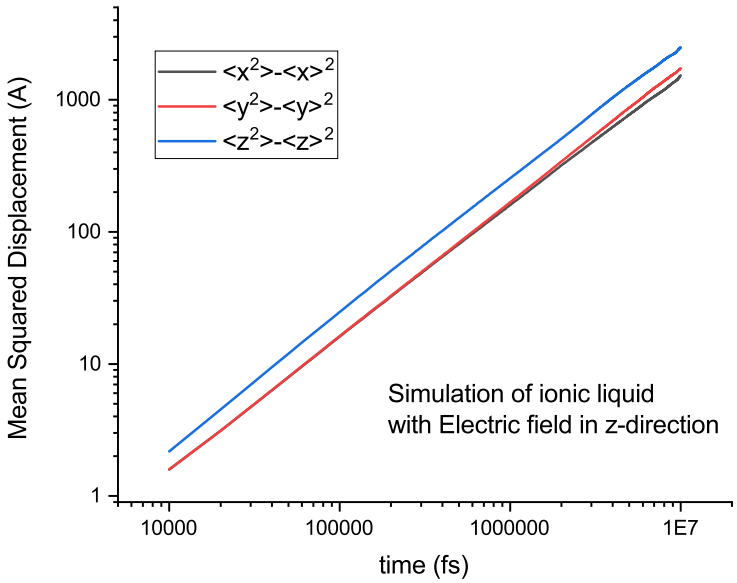

Figure 8: Drift-Corrected Mean Squared Displacement.

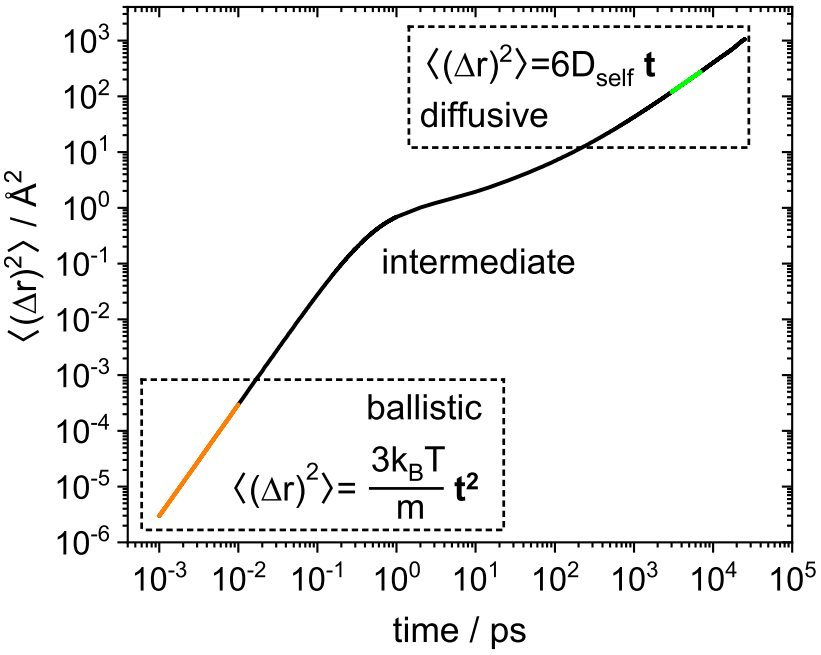

Figure 9: Mean squared displacement over 7 orders of magnitude, showing the distinct ballistic and diffusive regions. Taken from Reference [11].

2.7 Average distances

Calculates the average closest distance between a reference atom and an observed atom. Reference and observed atom can either be specific (i.e. a certain molecule_type_index and atom_index), or they can be of a certain type (such as 'H' or 'F'). Two modi are available: intermolecular or intramolecular distances. Also available are:

- exponentially weighted averaged distances (weighted with e^{-k*r})
- distances used in FFC theory, using eq (6) and (13) in the ESI of Reference [6].
- standard deviations of closest distances and exponentially weighted distances. We used these in Reference [1]

3 Getting Started

3.1 First Steps

Start by cloning the directory:

```
git clone https://github.com/FPhilippi/prealpha.git
```

In the directory 'prealpha', you can find the prealpha executable. You might need to add execution rights:

```
chmod +x prealpha
```

That's it already! You can now start using it. To get an idea of how the program is constructed, start with the examples, e.g. with

```
./prealpha Drudes_xyz.inp Drudes_vel.inp
```

This will give you an idea of the output and the structure of the program. You can also invoke the program without input file in the command line - it will assume that the input file is named 'general.inp'. The user interface of the program will be started if the current input file cannot be found.

If you want to compile prealpha yourself, please use a compiler that supports OpenMP. For the GNU fortran compiler, set the corresponding flag:

```
gfortran pre-alpha.f03 -fopenmp
```

The code is designed to also run without that library, but is not as nice. Naturally, compiling the code yourself is also the thing to do if the executable does not run on your architecture. Do let me know if you have problems with that.

3.2 Simple Examples

3.2.1 Drude Particles

For Drude particles to be recognised by prealpha, they have to belong to the same molecule type as their respective cores. This might need resorting of the trajectory. Two example trajectories for [BMIM][NTf2] are given in the folder 'Example_Drudes' - one with cartesian coordinates and one with velocities. Two general input files, 'Drudes_vel.inp' and 'Drudes_xyz.inp', can be found in the parent directory where the executable is located. Both can be invoked from the command line:

```
./prealpha Drudes_xyz.inp Drudes_vel.inp
```

The first input file assigns the Drude particles to their cores by checking closest distances. In the example the Drude-atom distances are between 0.001 and 0.09 Angström. The Drude assignment is not possible for a trajectory that contains velocities. Here, the Drude particles have to be assigned manually, as shown in 'molecular_VEL.inp' in the 'Example_Drudes' folder. The required section is conveniently printed when invoking 'show_drude' with cartesian coordinates, as in 'Drudes_xyz.inp'. Thus, the general approach is to make sure your Drude particles are grouped with the right molecule, then run 'show_drude' with a trajectory that contains cartesian coordinates, and either make changes to the suggested assignment or directly use it in your molecular input file - including for trajectories that contain velocities.

3.2.2 Estimating RAM

If you want to estimate how much RAM you need to store your entire trajectory, you can use the output of 'show_settings', such as in the example 'Drudes_xyz.inp'. Since sequential read of the trajectory is requested, the program only reads the first step (there aren't more in the example anyway), but still calculates the required RAM for the whole trajectory. In the corresponding molecular input file 'molecular.inp' in the folder 'Example_Drudes', you can see that 10000 steps were specified - this could be a trajectory of 10 ns, dumped every 1ps, for example. When prealpha is invoked with the example 'Drudes_xyz.inp', the following lines are shown in the settings section:

```
Memory requirement for storage of entire trajectory in RAM:
3x30208x10000x4Byte = 3.4GB (single precision)
```

Since there are 90624 coordinates to store, each of them taking up 4 Byte of RAM in single precision, this makes 3.4GB for the whole trajectory. Note that the whole LAMMPS trajectory would take up about 13GB of RAM![§]

[§]yes, I know, there is virtual memory and there is RAM, which is not the same.

3.2.3 The BMIMTFSI example trajectory (xyz)

This trajectory - again - contains just one timestep. It is in xyz format, but that's not a problem since you can specifiy the box boundaries manually, as is done in generalxyz.inp. This time, a warning will be printed, because prealpha is not sure whether there are coordinates or velocities stored in the trajectory due to the .xyz format. The input file is used for three simple examples:

- Calculating the radius of gyration
- Dumping dimer structures
- Getting intra- and intermolecular contact distances.

These analyses are in the generalxyz.inp input file in the parent directory. Invoke with:

```
./prealpha generalxyz.inp
```

The 'gyradius' keyword also calculates the ensemble average of the maximum distance of any atom in a specified molecule from the centre of mass. Note that the contact distances are very expensive to compute - that's why the code is parallelised, and also 'commented out' in the input file (Everything after a 'quit' statement is ignored). If you want more sophisticated distance estimates, have a look at the distance module which is a distince feature with its own input file.

Feel free to play around with the keywords! For example, instead of the cation-anion dimers, try dumping anion-anion pairs with 'dump_dimers T 1 1 1', or try to print a snapshot of the whole box. Maybe also wrap the trajectory before printing the snapshot!

3.3 The Input Trajectory File

The trajectory is expected to be in LAMMPS format, unless specified otherwise (Contact me if you need a different format to be read in). For each timestep, there is a header and a body. The header in LAMMPS format should look like this:

```
ITEM: TIMESTEP
0
ITEM: NUMBER OF ATOMS
20480
ITEM: BOX BOUNDS pp pp pp
0 63.9223
0 63.9223
0 63.9223
ITEM: ATOMS element xu yu zu
```

After this follows the body, consisting of one line per atom. Each line begins with the element label (e.g. 'C'), followed by three floating point (=real) number.

Depending on the type of analysis you need, these have to be either Cartesian coordinates or velocities. For coordinates, your LAMMPS input file should include something like:

```
dump TRAJECTORY all custom 1000 trajectory.lmp element xu yu zu
dump_modify TRAJECTORY element C F N O S C C C C H H N sort id
```

whereas for velocities, 'xu yu zu' has to be changed to 'vx vy vz'. To obtain sensible results, consistent ordering is imperative. This is the purpose of the second line given in the example above. Important final note: For performance reasons, the format is not checked during read!

3.4 Molecule Recognition Module

You will get fed up quickly with writing molecular input files, especially if you have many different trajectories. However, there is a way to automatically write those. You need to specify '-r' in the command line, immediately followed by your trajectory file. For example:

```
./prealpha -r./somedirectory/sometrajectory.lmp
```

or

```
./prealpha -rsometrajectory.lmp
```

... will try to recognise molecules in 'sometrajectory.lmp'. Only the very first step of the trajectory is used, and LAMMPS format is mandatory (the box volume is required for Periodic Boundary Conditions). If you have very big molecules the molecule recognition might fail, because there is this one variable that speeds up things. Let me know if you have issues and I'll change it. However, keep in mind that I originally made the recognition module for debug purposes only - it is not even remotely optimised, and eats a lot of virtual memory. TRAVIS is much better written in that respect.[3] Also, in later versions, you can leave a space:

```
./prealpha -r sometrajectory.lmp
```

The molecular input files are written in the same directory as the trajectory, i.e. in './somedirectory/' in the first example. That way, you can leave prealpha in one place.

3.5 High Performance Computing

If you're running prealpha on your favourite supercomputer, you might want to make use of parallelisation. It is strongly advised to load the trajectory into RAM, as almost all involved calculations become prohibitively slow if the sequential read is used. prealpha requires an absolute minimum amount of RAM to store the trajectories - see the 'estimating RAM' section in 'Examples' in this wiki on a short guidance how to calculate the required RAM.

Finally, when you have loaded your trajectory in the RAM, you might want to try the parallelisation. Let's say you work on a node with 48 cores, and you have four trajectories to analyse. Before invoking prealpha (e.g. in your PBS script), you should set the corresponding environment variable to the desired number of cores, in our case 48/4=12:

```
export OMP_NUM_THREADS=12
```

It is good practise to always set this variable, and in addition, use the keywork 'set_threads'. If you're lazy, you can always add 'set_threads 0' to your general input file, as this will lead to prealpha taking all the threads specified in OMP_NUM_THREADS.[¶]

After that, you can invoke four instances of prealpha, and - if you want - redirect their output to certain output files. To achieve this, you can use the '-d' command line argument of prealpha - if any of the command line arguments is -dsomestring, then the output from unit 6 (standard output) is redirected to a file named 'somestring'. In our example case, you could use this in your PBS / bash script:

```
./prealpha general1.inp -d./output1/FORTRANLOG &
./prealpha general2.inp -d./output2/FORTRANLOG &
./prealpha general3.inp -d./output3/FORTRANLOG &
./prealpha general4.inp -d./output4/FORTRANLOG &
wait
```

... which redirects the output into files called 'FORTRANLOG' in the folders output1 to output4. Ideally, these would be specified as output directory in the input files general1.inp to general4.inp, respectively. Of course you can also run the analysis one after another, on all 48 cores each. This will be a bit more efficient if the four jobs are very different:

```
export OMP_NUM_THREADS=48
./prealpha general1.inp -d./output1/FORTRANLOG
./prealpha general2.inp -d./output2/FORTRANLOG
./prealpha general3.inp -d./output3/FORTRANLOG
./prealpha general4.inp -d./output4/FORTRANLOG
```

[¶]You can also use set_threads_simple

4 Input Files

The general and molecular input files are required in any case by the main program. Depending on the desired calculation, a separate input file might be required.

4.1 General Input File

The 'general.inp' file is the main input file, located in the same folder as the executable. It is possible to specify other names for the general input file as command line arguments. When multiple general input files are specified, they will be invoked subsequently. If the file / one of these files isn't found, then the program switches to user input. It is read line wise, with the first 5 lines being reserved (and strictly fixed). The content of these lines is, in this order:

1. the filename of the trajectory, e.g. 'trajectory.lmp'
2. the name of the molecular input file, e.g. 'mymolecule.inp'
3. Path to the trajectory
4. Path to the input files other than the general and molecular input files.
5. Output folder path

Path names have to be enclosed in quotes. The body of 'general.inp' is read line-wise, and the program finishes when either 'quit' or the end of file is encountered. Each line contains a switch or keyword, followed by an argument (if required). Only the necessary information is read from any line, with the rest being ignored. Be aware that keywords affect only the lines below them. This is with the exception of sequential_read, trajectory_type, unwrap_trajectory and wrap_trajectory. These latter three act on the whole analysis, no matter where specified. Only their first occurence matters - everything afterwards is ignored. An incorrectly formatted 'general.inp' is not tolerated (read the error messages). For many switches, a simple mode that doesn't require additional input is available. The simple mode is requested by appending '_simple' to the switch, e.g. 'gyradius_simple'. Available switches are: (case-sensitive, everything is lowercase)

- 'sequential_read': If true 'T', then the trajectory is read line by line. This is slow, but requires only the minimum amount of RAM. Not recommended for parallelised analyses like mean-squared displacement and VACFs. If false 'F', then the whole trajectory is read into RAM. This is the first switch that affects every line, not just the ones after it.
- 'trajectory_type': Expects either 'xyz' or 'lmp' as string input. This is the second switch that affects every line, not just the ones after it.

- 'wrap_trajectory': Expects one logical. If 'T', then molecules are wrapped into the box. (based on their centre of mass. Might not be sensible for some analyses.) This is the third switch affecting every line.

- 'unwrap_trajectory': Expects one logical. If 'T', then molecules are unwrapped, i.e. the molecules are translated to minimise the jump distance between subsequent timesteps. 'unwrap_trajectory' is only available with 'sequential_read F'. unwrapping is performed based on centre of mass, only works if jumps are smaller than half the box length. This is the fourth switch affecting every line.

- 'parallel_operation': Turns parallelisation on (T) or off (F). Parallelisation is only available with 'sequential_read F'

- 'set_threads': (simple mode available) Sets the number of threads to use. 'set_threads_simple' uses all available threads.

- 'error_output': Turns error output on (T) or off (F).

- 'time_scaling': Takes an integer value, by wh ich the timestep is multiplied. For example, specify 'time_scaling 1000' if your trajectory is dumped every 1000 fs, and your output time unit will be fs as well.

- 'set_prefix': The specified prefix is prepended to the output files. Useful if, for example, the dihedral analysis is specified multiple times.

- 'dump_example': Writes an xyz file of every specified molecule type into the output folder. Can be used to extract the atom numbers for the dihedral analysis.

- 'dump_snapshot': (simple mode available) Expects an integer and a logical and dumps the specified timestep as .xyz file. If the logical is 'T', then every molecule is written into a separate file.

- 'dump_split': (simple mode available) Splits the trajectory into separate files for every molecule type (centred to centre of mass!). Expects two integers: the first timestep and the last timestep.

- 'dump_single': Writes a trajectory containing just one single molecule. This keyword expects a logical, followed by four integers in the same line: If the logical is 'T', then the molecule is centred to its centre-of-mass. The first and second integers specify the first and last timestep to write. The third and fourth integers are the molecule type index and the molecule index, respectively.

- 'dump_full_gro','dump_full_xyz','dump_full_lmp': Writes the whole trajectory to GROMACS, xyz, or LAMMPS format, respectively.

- 'slab_x', 'slab_y', 'slab_z': Any of these commands writes an xyz file containing a slab where the x, y, z directions are normal to the slab plane, respectively. The slab is positioned in the middle of the box, and wrapping is automatically performed molecule wise. This switch requires two integers: First the timestep to export, and second the molecule type index. The molecule type index can also be -1, in which case all molecule types are considered. Note that any molecule is exported which has any atom with a position higher than the lower boundary plane AND lower than the higher boundary plane - in the current implementation, both boundary planes are set to the middle of the box boundaries in the respective dimension.

- 'contact_distance': (simple mode available) Reports the smallest intra- and intermolecular distances and the largest intramolecular distance. This keyword expects two integers as input: the timestep to analyse and the molecule type index. If a molecule type index of 0 is specified, then all molecule types are considered.

- 'dump_cut': Like dump_single - but the surrounding molecules are also written. This keyword expects a logical, followed by four integers and one real in the same line: If the logical is 'T', then the molecule is centred to its centre-of-mass in every step. The first and second integers specify the first and last timestep to write. The third and fourth integers are the molecule type index and the molecule index, respectively. The real number defines the cutoff for centre-of-mass distance for exporting molecules. Note that the properly wrapped mirror images of the closest encounters are given.

- 'dump_dimers' Dumps the closest molecule of type X around all molecules of type Y for a certain timestep. Expects a logical, followed by three integers. If the logical is (T), then the output is combined in a single 'trajectory'-like xyz file. Otherwise (F), one output file is written per dimer. The first integer is the timestep. The following two integers are the types Y and X, respectively

- 'dump_neighbour_traj': (simple mode available) Dumps the N closest molecules of type X around a certain molecule M of type Y as trajectory. This keyword expects six integers: The first and second integers specify the first and last timestep to write. The third and fourth integers are the molecule type index Y and the molecule index M, respectively. The fith integer is the integer of the neighbour molecule to consider The last integer is the number of neighbours to write.

- 'cubic_box_edge': this keyword expects two real values, the lower and upper bounds of the simulation box. i.e. 'cubic_box_edge 0.0 100.0' corresponds to a cubic box with side length 100.0 Angströms useful if e.g. dump_cut is used with a xyz trajectory.

- 'convert': (simple mode available) converts the given trajectory to a centre-of-mass trajectory (per specified molecule type). i.e. only the centres of mass for the molecules are printed instead of the atoms. This keyword expects a logical. If (T), then a new, modified molecular input file is written as well.
- 'convert_coc': converts the given trajectory to a centre-of-charge trajectory (per specified molecule type). i.e. only the centres of charge for the molecules are printed instead of the atoms. This keyword expects a logical. If (T), then a new, modified molecular input file is written as well. Requires atomic charges to be properly initialised.
- 'temperature': (simple mode available) Computes the instantaneous temperature of a particular molecule type. This keyword expects exactly three integers: The molecule type index, and the range of analysis, given as first step and last step. If a molecule type index of 0 is specified, then all molecule types are considered.
- 'drude_temp': (simple mode available) Computes Drude, centre of mass, and total temperature of the whole box. This keyword computes equation (13), (14) and (15) in [13]. Support of Drude particles requires the Drude particles to be read in manually, since the automatic Drude particle assignment is only available for position trajectories. This keyword expects exactly two integers: The range of analysis, given as first step and last step.
- 'remove_drudes': (simple mode available) writes a new trajectory, with Drude particles merged into their respective cores. (requires assigned Drude particles, either manually or automatically) This keyword expects exactly two integers: The range of analysis, given as first step and last step.
- 'remove_cores': (simple mode available) writes a new trajectory only with Drude particles minus the positions of their respective cores. (requires assigned Drude particles, either manually or automatically) This keyword expects exactly two integers: The range of analysis, given as first step and last step.
- 'gyradius': (simple mode available) Computes the ensemble averages and standard deviations of radius of gyration, radius of gyration squared, and maximum distance of any atom in a molecule from its centre of mass. This keyword expects exactly three integers: The molecule type index, and the range of analysis, given as first step and last step. If a molecule type index of 0 is specified, then all molecule types are considered. The result will look something like this:

```
<gyradius>  0.3326E+01, stdev = 0.870E-01
<gyrad**2>  0.1107E+02, stdev = 0.579E+00
<maxdist>   0.6159E+01, stdev = 0.252E+00
```

The cornered brackets denote the ensemble average, the results are reported in the same unit as the input trajectory - ideally Ångström. The squared radius of gyration and radius of gyration of a molecule are defined in Equation 2 and Equation 3, respectively. Here, m_i is the mass of atom i in the molecule, and r_i is the distance of the atom from the centre of mass of the molecule. The sums run over all atoms in the molecule.

$$Rgy^2 = \frac{\sum_i m_i r_i^2}{\sum_i m_i} \tag{2}$$

$$Rgy = \sqrt{\frac{\sum_i m_i r_i^2}{\sum_i m_i}} \tag{3}$$

- 'jump_velocity': (simple mode available) Computes a histogram / probability distribution of jump velocities. This keyword expects exactly three integers: The molecule type index, the range of analysis, given as first step and last step, and the maximum jump length / shift given as number of timesteps. If a molecule type index of 0 is specified, then all molecule types are considered.
- 'show_settings': Writes settings and useful information to the standard output
- 'print_atomic_masses': Writes atomic masses to the standard output, using the molecular input file format.
- 'print_atomic_charges': Writes atomic charges to the standard output, using the molecular input file format.
- 'print_dipole_statistics': outputs average/minimum/maximum/standarddev of dipole moment for the first timestep. for charged molecules, the vector from center of mass to center of charge is used.
- 'show_drude': Writes detailed current information about Drude particles.
- 'switch_to_com' Irreversibly switches to the barycentric reference frame. i.e. for all analyses below this keyword, the box' centre-of-mass is removed in every step. (cannot be turned off again, until the program switches to the next general input file)
- 'conductivity_simple' This is a special keyword that computes the overall electrical conductivity of the whole system. It uses the autocorrelation module, but is much faster than the CACFs.
- 'verbose_output': Turned on (T) by default. If (F), then only very limited output is obtained.
- 'time_output': Turns the timing on (T) or off (F).
- 'quit' Terminates the analysis. Lines after this switch are ignored.

These keywords require separate input files (explained below):

- 'conductivity' (requests feature 'cacf components' or 'conductivity_simple')
- 'velocity' (requests feature 'vcf components')
- 'rmm-vcf' (requests feature 'Relative Mean Molecular Velocity Correlation Coefficients')
- 'diffusion' (requests feature 'Mean Squared Displacement') (two simple modes available, 'diffusion_simple' and 'alpha2_simple')
- 'dihedral' (requests feature 'Dihedral Conditions')
- 'reorientation' (requests feature 'reorientational time correlation')
- 'distribution' (requests feature 'polar/cylindrical distribution function) (simple mode available - 'charge_arm_simple' to calculate the charge arm distribution, 'clm_simple' for the charge lever moment distribution, and 'distribution_simple' to calculate sum rules and the coulomb interaction energy)
- 'distance' (requests feature 'Average distances') (simple mode available)

4.2 Molecular Input File

This file contains information about the system, located in the same folder as the executable. The first line is the number of timesteps, followed by the number of molecule types in the second line. For every molecule type, the following information is read: Charge - Number of atoms per molecule - number of molecules. The program expects as many lines as there are molecule types. Following this fixed section, the rest of the input file is read (Until either a 'quit' statement or the end of file are encountered). In this free-format section, the following optional subsections can be placed:

- 'default_masses': this keyword triggers the specification of custom default masses. it expects an integer, which is the number of subsequent lines to read. This is available for single lowercase letters (a,b,c,...,z) and element names. (Including 'X' and 'D', which are treated as Drude particles) If e.g. the trajectory contains an anion of mass 123.4, abbreviated as 'a', and a cation of mass 432.1, abbreviated as 'c', then this section should be added:

```
masses 2
a 123.4
c 432.1
```

  Furthermore, the support of Drude particles can be turned on by adding:

```
masses 1
X 0.4
```

Note that Drude particle masses are subtracted from N,O,C,S,P,Li,F - but not Hydrogen. This keyword changes the defaults, i.e. ALL atoms of type a,X,P,... see also:

- 'atomic_masses': This keyword triggers the specification of custom atomic masses. it expects an integer, which is the number of subsequent lines to read. each line must have two integers and one real: molecule type index, atom index, and atomic mass. This allows the user to specify atomic weights, which can be misused two work with centres of charge rather than centres of mass.[‖] The order is important:

  1. The program starts with its own defaults, such as 12.011 for carbon.
  2. If necessary, these values are changed by 'default_masses' - this changes all masses for one element type.
  3. Any positive Drude mass 'X' or 'D', if present, is subtracted from N,O,C,S,P,Li,F.
  4. After that, masses of particular atoms are overwritten by 'atomic_masses'.

- 'default_charges' and 'atomic_charges' These two keywords can be used to specify atomic charges. Their syntax follows that of the keywords 'default_masses' and 'atomic_masses', respectively (With the exception that lowercase letters are not accepted). The order is as follows:

  1. The program starts with its own default charge - 0.0 for every atom.
  2. If necessary, these values are changed by 'default_charges'
  3. Charges of particular atoms are overwritten by 'atomic_charges'.

- 'constraints': This keyword triggers the specification of custom constraints. It expects an integer, which is the number of subsequent lines to read. Each of these subsequent lines has to contain two integers: First, the molecule type index, and second, the number of constraints. This influences how temperature is computed.

- 'Drudes': This keyword is used to manually assign Drude particles to their respective core. It expects an integer, which is the number of subsequent lines to read. Each Drude particle is assigned by giving three integers (per line): The molecule type index - atom index core - atom index Drude.[**]

[‖] Note that the emphasis is on 'misused'. It does make sense in some cases, for example for the Coulomb interaction energy, which is more accurate if the centres of charge are used rather than the centres of mass. However, modules that *need* atomic charges (in addition to atomic masses) will make use of the appropriate keyword, 'atomic_charges'

[**] see also 'show_drude', which prints this input section. If you have Cartesian coordinates, then the Core-Drude assignment will be automatic - I did not have problems with that so far.

4.3 Relative Mean Molecular Velocity Correlation Input File

The two molecules to correlate have to be given in the first line. 'rmm-vcf' is given in the second line, indicating the type of analysis. Switches are read from the following lines. Available are:

- 'tmax': Expects an integer, which is then taken as the maximum number of steps into the future for the autocorrelation function (the shift, so to say).
- 'skip_autocorrelation': If yes (T), then only the cross-contributions are calculated.
- 'sampling_interval': Expects an integer. Every so many steps will be used as origin to compute self-contributions. These are usually computationally more expensive, but need less averaging. Note that the printed tcf will always have the same time resolution as the trajectory.
- 'quit' Terminates the analysis. Lines after this switch are ignored.

4.4 Custom Velocity Correlation AND CACF Components Input File

The first line gives the operation mode, followed by the number of custom components. The operation mode is either 'vcf' or 'cacf'. The custom components are specified below, one per line. Each line must contain, in this order:

1. the molecule type index of the first, reference molecule
2. the molecule type index of the second, observed molecule
3. 'T' if self contributions are to calculate, and 'F' for distinct contributions.

Thus, the following 7 lines specify every unique vcf in a system with two constituents:

```
vcf 6
1 1 T
1 1 F
1 2 F
2 1 F
2 2 T
2 2 F
```

(note that '1 2 F' and '2 1 F' are redundant and will/should give the same value)

Switches are then read from the following lines. Available are:

- 'tmax': Expects an integer, which is then taken as the maximum number of steps into the future for the autocorrelation function (the time shift).

- 'sampling_interval': Expects an integer. Every so many steps will be used as origin of the correlation functions. Note that the printed tcf will always have the same time resolution as the trajectory.
- 'quit' Terminates the analysis. Lines after this switch are ignored.

A rule of thumb as final remark: everything with an 'F' will be very, very, very slow. This is because the double sum over every distinct particle is evaluated. It is possible (and a lot faster) to just calculate the overall electrical conductivity. To request this, just put 'conductivity' in the first line. Also, have a look at the cross-correlations obtained from mean squared displacements, i.e. the Einstein Relation equivalent of the RMM-VCFs.

4.5 Diffusion Input File for Self Diffusion

The first line contains the expression 'msd', followed by the number of projections N.[††] The latter are read from the following N lines. The format of each line is: x - y - z - number of the molecule type. For the 'standard' 3D diffusion of molecule type 2, the line would thus be '1 1 1 2'. After the projections have been specified, switches can be specified in an arbitrary order. Available are:

- 'tmax': Expects an integer, which is then taken as the maximum number of steps into the future for the mean squared (or whatever exponent you want) displacement.
- 'tstep': The given integer is taken as the step size. i.e. if 'tstep 10' is specified, then only shifts by 1,10,20,...,tmax are computed.
- 'print_verbose': If yes (T), then the detailed drift is printed, too.
- 'exponent': Expects an integer, which is used as exponent in the displacement. Thus, 'exponent 2' is the normal MSD, but others can be used, too.
- 'quit' Terminates the analysis. Lines after this switch are ignored.

4.6 Diffusion Input File for Cross Diffusion

The first line contains the expression 'cross', followed by the number of projections N. The latter are read from the following N lines. The format of each line is: x - y - z - number of the first molecule type - number of the second molecule type. To get the cross diffusion between the first two molecule types, the line would thus be '1 1 1 1 2'. After the projections have been specified, switches can be specified in an arbitrary order. Available are:

- 'tmax':Expects an integer, which is then taken as the maximum number of steps into the future for the mean squared displacement.

[††]'alpha2' can also be used instead of msd. note that while projections other than 1-1-1 are allowed, they will give a wrong alpha2.

- 'tstep': The given integer is taken as the step size. i.e. if 'tstep 10' is specified, then only shifts by 1,10,20,...,tmax are computed.
- 'exponent': Expects an integer, which is used as exponent in the displacement. Thus, 'exponent 2' is the normal MSD, but others can be used, too.
- 'quit' Terminates the analysis. Lines after this switch are ignored.

4.7 Distribution Input File

(Simple modes available: 'charge_arm_simple' to calculate the charge arm distribution, 'clm_simple' for the charge lever moment distribution, and 'distribution_simple' to calculate sum rules and the coulomb interaction energy.) The first line contains the expression 'cdf', 'pdf' or 'charge_arm', followed by the number of references N. 'cdf' requests the cylindrical distribution function, 'pdf' requests the polar distribution function. 'charge_arm' requests a polar distribution function of the charge arm (ions) or dipole moment (neutral molecules). The references are read from the N lines following the first line. The format of each line is: x - y - z - number of reference molecule type - number of observed molecule type. For molecule type 1 around 2 relative to z-direction, the line would thus contain '0 0 1 2 1'. a 'zero' reference vector, i.e. '0 0 0', triggers the randomisation of the reference vector. If the molecule type of the observed molecule is -1, then *all* molecule types are used. Note that for 'charge_arm', only one molecule type is required - no observed molecule type is required, only the reference type. The charge arm pdf is NOT corrected for azimuthal or radial parts - only polar. After the projections have been specified, switches can be specified in an arbitrary order. Available are:

- 'bin_count': Expects an integer, which is then used as bin count for both independent variables.
- 'bin_count_a': Expects an integer, which is then used as bin count for variable a).
- 'bin_count_b': Expects an integer, which is then used as bin count for variable b).
- 'maxdist': Expects a real value, which is taken as the cutoff distance of molecule pairs to be considered.
- 'maxdist_optimize': sets 'maxdist' to half the box size where available. doesn't need additional input values. for charge arm and dipole moment analyses, 'maxdist' will be set to the maximum value in the first timestep. (considering all molecule types specified as references, rounded to 1 digit)
- 'subtract_uniform' If yes (T), then the uniform density / radial distribution function is subtracted. Only available for the polar distribution function.

- 'weigh_charge' The charges of observed molecules are added to the distribution histogram, rather than unity.
- 'normalise_CLM' The charge arm is divided by $M \cdot Rgy^2$ (charge lever moment correction).[8] Here, M is the mass of the molecule, and Rgy is the radius of gyration. Only available for the charge arm distribution function.
- 'center_of_charge' Uses centres of charge instead of centre of mass. Needs atomic charges to be specified. For molecules with a total charge of zero, the dipole moment will be used, which might yield unexpected results.
- 'sampling_interval' Expects an integer. Every so many steps are used for the analysis.
- 'quit' Terminates the analysis. Lines after this switch are ignored.

4.8 Dihedral Input File

The molecule type index (= the molecule to observe) is given in the first line. The expession 'dihedrals', followed by the number of dihedral conditions, in the 2nd line. For every dihedral condition follows one line, giving the atoms (in that molecule) which are part of the dihedral, as well as the lower and upper bound. '1 2 3 4 0.0 90.0' thus means that dihedral 1-2-3-4 has to be between 0 and 90 degrees. Important: dihedrals are defined from 0.0° to 360.0°. ALL specified conditions have to be fulfilled simultaneously for the h operator to become true. After the condition section, the following switches may follow:

- 'tmax': Expects an integer, which is then taken as the maximum number of steps into the future for the intermittent binary autocorrelation function.
- 'export': Requires one integer (the index of the molecule) as input. All specified dihedrals for this particular molecule will be exported in an output file. Note that 'export' can be specified more than once!
- 'fold': If true (T), then apart from the dihedrals being in the range a to b, also check for the range (360-b) to (360-a).
- 'dump_verbose': If true (T), then also report PES subset population and $\langle h \rangle$ as a function of the timestep. Also check for the range (360-b) to (360-a).
- 'skip_autocorrelation': If true (T), then the actual autocorrelation analysis is skipped. This is useful if only the PES is required.
- 'bin_count': Sets the bin count to the specified integer. e.g. 'bin_count 36' equals to binning in steps of 10°.
- 'jump_analysis' Performs an analysis of average jump velocities as a function of: a) The number of changes between fulfilment/non-fulfilment of the dihedral condition b) The share of fulfilled dihedral conditions over

the specified jump time. Requires one integer, i.e. the desired jump time (in number of timesteps). It is possible to request multiple jump analyses in one dihedral input file.

- 'quit' Terminates the analysis. Lines after this switch are ignored.

4.9 Reorientation Input File

The molecule type index (= the molecule to observe) is given in the first line. The expession 'reorientation' in the second line to request the appropriate analysis. For the reorientation analysis, the user has the choice between two vectors:

1. The vector from a base fragment to a tip fragment. Both *must* be defined as outlined below.

2. the charge arm or dipole moment vector. This required atomic charges (cf. molecular input file)

Regarding method 1): A fragment is defined by the expression 'base' or 'tip', followed by the number of atoms in this fragment. The immediately following line must contain a list of the atom indices in this fragment. For example, these lines define atom 16 as the base fragment and atoms 1, 3 and 4 as the tip fragment:

```
base 1
16
tip 3
3 1 4
```

The two fragments must appear before the quit statement (if applicable). Method 2) only requires 'charge_arm' or 'dipole', but no further input (apart from the atomic charges). The following switches may be used as well:

- 'tmax': Expects an integer, which is then taken as the maximum number of steps into the future for the time correlation function.

- 'legendre': Expects an integer, which defines the order of the legendre polynomial to use.

- 'sampling_interval': Expects an integer. Every so many steps will be used as starting point for the tcf. Note that the printed tcf will always have the same time resolution as the trajectory.

- 'export': Requires one integer (the index of the molecule) as input. The orientation evolution for this particular molecule will be exported in an output file, containing timestep, unit vector, vector length, angle and $P_l[u(0)u(t)]$

4.10 Distance Input File

The first line must state the type of analysis, i.e. either 'intramolecular' or 'intermolecular', followed by the number of subjobs as an integer. The following lines, one for each subjob, describe which atoms are reference and observed atoms. For intramolecular analysis, you need to specify one of these for every subjob:

- Three integers: the molecule type index and the two atom indices for reference and observed atom, for example '3 1 2' calculates closest intramolecular distances for the atoms with indices 1 and 2 in molecule type 3.
- two integers and an element name, acting as wildcard. The program will automatically consider all atoms of that type, for example '3 1 H' uses all H atoms in molecule type 3 as observed atoms.
- two element names, such as 'H H'. Note that this type of analysis averages over all possible intramolecular combinations, if you have more than one molecule type containing hydrogen atoms then you will obtain an average over all of these.

The same principles apply for intermolecular distance analyses, only that the second molecule type index needs to be given:

- Four integers: the molecule type index and atom index for the reference atom and the molecule type index and atom index for the observed atom. Note that even if the molecule type indices are the same, the analysis is still intermolecular - and thus makes only sense if there are at least two molecules of that type.
- two integers and an element name, acting as wildcard. The program will automatically consider all atoms of that type, for example '3 1 H' uses all H atoms in all molecule types as observed atoms.
- two element names, such as 'H H'. Note that this type of analysis averages over all possible intermolecular combinations.

After the subjob section, the following switches may follow:

- 'quit' Terminates the analysis. Lines after this switch are ignored.
- 'maxdist': Expects one real number, which is the maximum distance / cutoff to consider.
- 'maxdist_optimize': sets the cutoff to half the box size.
- 'nsteps': Followed by one integer, which is interpreted as the highest timestep number to consider.
- 'sampling_interval': Expects one integer - the sampling interval, i.e. every this many steps of the trajectory will be used.

- 'ffc': expects a logical ('T' or 'F'). If 'T', then the average distances used for FFC are calculated, i.e. dHH and rHH depending on the operation mode. Note that this average does not converge, which is not my fault.
- 'calculate_exponential': expects a logical ('T' or 'F'). If 'T', then the weighted average distances are also calculated. the weights are exp(-kr), where r is the distance and k is a constant defaulting to 1.
- 'exponent': sets the exponent 'k' for the exponentially weighted average.
- 'standard_deviation': expects a logical ('T' or 'F'). If 'T', then the standard deviation is - where reasonable - calculated.

It is easiest to just use the simple mode by adding "distance_simple" to the general input file. This will prepare input files and run the analysis on the fly, assuming that only Hydrogen and Fluorine - where present - are the nuclei of interest. The typical output looks something like this:

```
Done with intramolecular distance calculation. Results:
  Subjob #1 out of 1:
    30 H atoms -> 30 H atoms.
    Used 15360 reference atoms per step.
    average closest distance: 1.734
    standard deviation: 0.053
    exponentially weighted distance: 2.705
    standard deviation: 0.211
    FFC distance rHH: 2.773
    exp(-kr) weighted FFC distance r'HH: 2.015
    Number of averages for FFC: 0.445E+07
```

For the simple analysis, all other hydrogen atoms in the molecule (intramolecular) or all other hydrogen atoms in other molecules (intermolecular) are considered. The "average closest distance" finds the closest other hydrogen atom in the set for each of the hydrogen atoms, and then takes the average of the closest distances. The exponentially weighted average uses a weighing function e^{-kr} to calculate the weighted average. For the use with fast field cycling (FFC), Equation 4 (intramolecular) and Equation 5 (intermolecular) are used.

$$r_{HH} = \left(\frac{\sum_{i \neq j} w_{ij} r_{ij}^{-6}}{\sum_{i \neq j} w_{ij}} \right)^{-1/6} \tag{4}$$

$$d_{HH} = \left(\frac{\sum_{i \neq j} w_{ij} r_{ij}^{-3}}{\sum_{i \neq j} w_{ij}} \right)^{-1/3} \tag{5}$$

The weighing factor w_{ij} is either 1 (rHH and dHH in the output above) or $w_{ij} = e^{-kr_{ij}}$ (r'HH and d'HH in the output above).

5 Research Examples

5.1 Research Example 1: Ammonium Ionic Liquids

The required input files for this example[12] can be found here: `https://github.com/FPhilippi/prealpha/tree/master/Research_Example_1` In the example, the analysis is started by submitting 'invoke_prealpha_simple.pbs' to the PBS queuing system of a high performance computing facility. prealpha will then perform the analyses in 'general.inp'. These are:

- radius of gyration, maximum distance of any atom in a molecule from its centre of mass (line 17)
- a charge arm distribution (line 21), as described in the additional input file 'charge_arm_cation.inp'
- histogram of the dihedral angles in the anion (the two C-S-N-S dihedrals) and the cation (all 9 backbone dihedrals) (line 24 and 26)
- the Coulomb energy integral (line 28). This will also produce a charge weighted polar distribution function of all ions around a central ion (file ends in '_pdf') and the sum rules (file ends in '_numberintegral')
- mean squared (line 31) and mean quartic (line 33) displacements, from which diffusion coefficients and alpha2 parameters can be obtained.

Note that in the corresponding molecular input file 'molecular.inp', the atomic charges need to be specified for the charge arm analyses to make sense. It is advisable to perform a separate analysis using the centre of charge instead of the centre of mass for the Coulomb energy integrals. To this end, the atomic charges are simply passed to prealpha as masses, as shown in 'molecular_COC.inp' and 'general_COC.inp'.[‡‡]

5.2 Research Example 2: Charge Transfer and Polarisability

Most inputs - such as those for the Coulomb energy integrals - were already part of the Research Example 1, *cf.* also Figure 4 and Figure 6. The additional analyses in this example[10] can be performed with the input files in section 3.2.1, *cf.* the folder 'Example_Drudes' and the general input files 'Drudes_vel.inp' and 'Drudes_xyz.inp' in the parent directory. The following two commands are relevant in 'Drudes_xyz.inp', both compatible with sequential read:

```
remove_drudes 1 1
remove_cores 1 1
```

[‡‡]This workaround is now no longer necessary - see 'center_of_charge' in subsection 4.7

The first command (line 9) collapses Drude particle and Drude core to just one real atom. This is a very useful processing step to make trajectories with Drude particles digestible to other software. It is also worth pre-processing the trajectory in this way before subsequent analyses with prealpha which do not require the Drude particles, this gives better performance and needs less memory.

The second command (line 10) prints Drude particle positions relative to their core. This might not seem very useful, but can be used in combination with TRAVIS[3,2] to generate a Power spectrum of just the Drude internal motions, Figure 10.

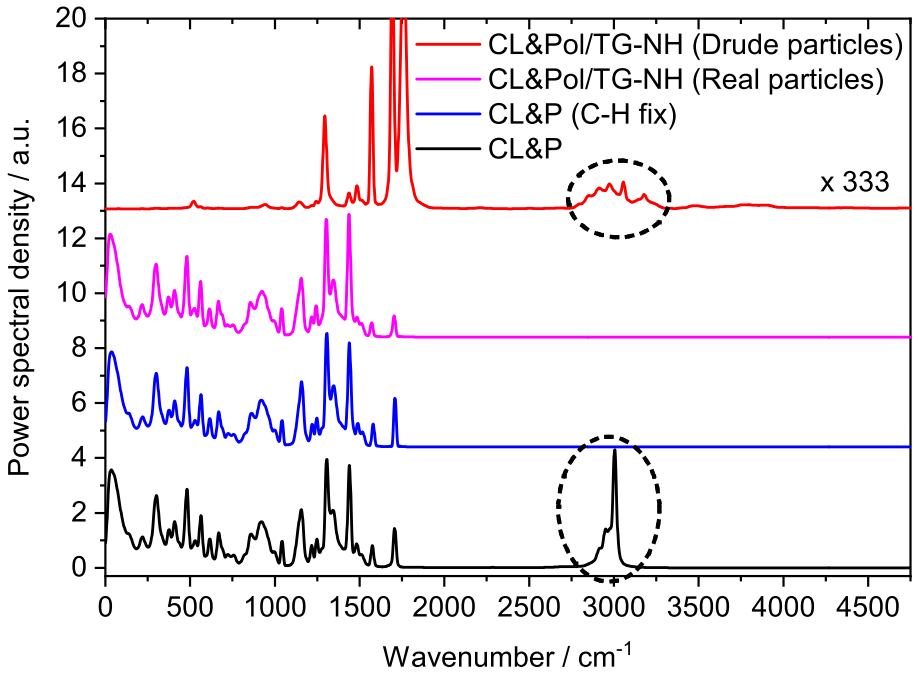

Figure 10: Power spectra for different simulation setups (vertically offset). red = internal motions of Drude particles, multiplied with 333 (*via* remove_cores). The temperature of real degrees of freedom was 333 K, while Drude internal motions were kept at 1 K. pink = motions of real degrees of freedom in the same simulation (*via* remove_drudes). Blue and Black are non-polarisable simulations with and without C-H constraints, respectively. The dashed black circles show undesired overlap of high frequency oscillations. Adapted from Ref. [10] with permission from the PCCP Owner Societies.

The 'drude_temp' command in the 'Drudes_vel.inp' input file (line 7) is useful to check thermostatting. The trivial input example should generate the following output:

```
TCM: 360.9 K (1024)
TR:  286.7 K (61440)
TD:    1.1 K (9728)
```

The degrees of freedom are given in brackets; TCM / TR / TD are equation (13) / (14) / (15) in reference [13], respectively. An example is shown in Figure 11.

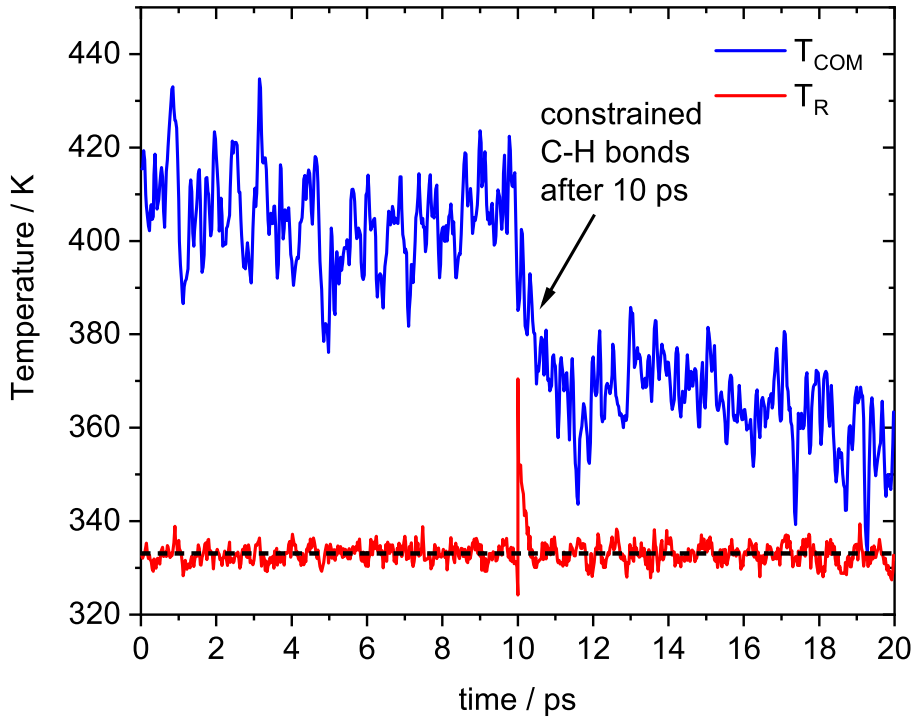

Figure 11: The temperature of the centre of mass motions (TCOM=TCM) may be off if undesired heat transfer from Drude to real degrees of freedom violates equipartition. The signature of this heat transfer is not visible in the temperature of the real degrees of freedom TR, which is what a primitive temperature control algorithm might use. Adapted from Ref. [10] with permission from the PCCP Owner Societies.

5.3 Research Example 3: Advanced Correlation Functions

Here we investigated the dynamics in ionic liquids with different anions (The input files etc are deposited in the reference).[11] Figure 12 shows the decorrelation of several time correlation functions over 7 orders of magnitude from 1 fs to 10 ns. Two trajectories were used, one dumping with a resolution of 1 fs (to calculate the TCFs up to 1ps), and a longer one dumping every 1 ps (for the TCFs >1 ps). Dynamical heterogeneity was investigated using the α_2 parameter, Figure 13. The intermittent dihedral autocorrelation function is shown too; these two phenomena (dynamical heterogeneity and cis-trans interconversion) occur on comparable timescales. Finally, we also calculated the diffusion coefficients by fitting the diffusive region of the mean squared displacements, Figure 14. The diffusion coefficients were determined independently for x/y/z directions, the average is identical to the diffusion coefficient determined from the three dimensional mean squared displacement. However, using the three dimensions separately, an error estimate is possible (see the error bars).

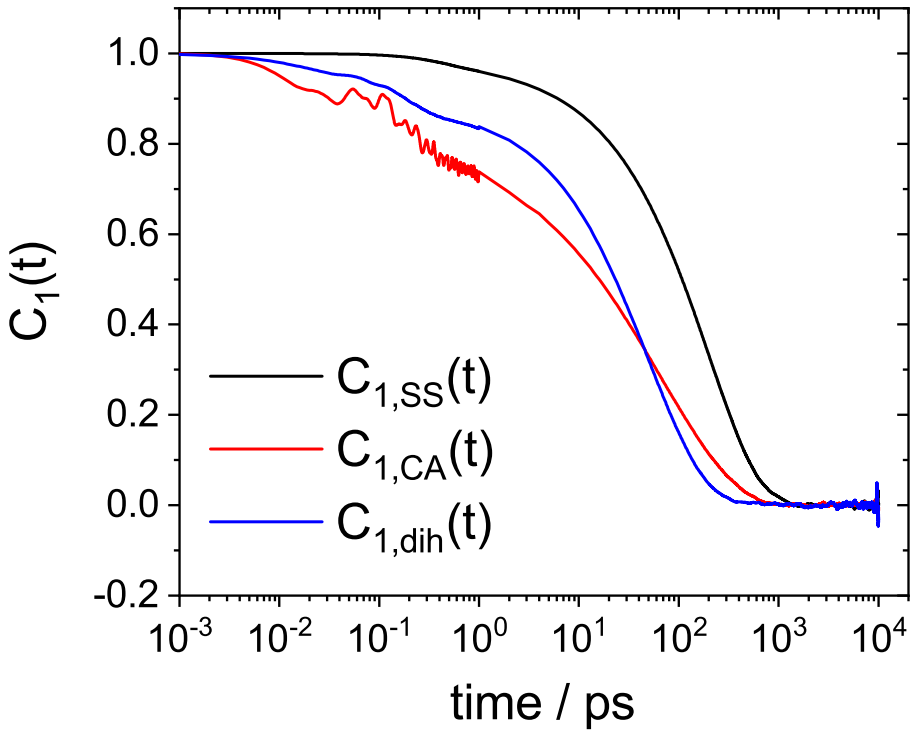

Figure 12: Different time correlation functions of the $[NTf]_2^-$ anion. Here, $C_{1,SS}$ is the vector reorientation function using the vector connecting two sulphur atoms; $C_{1,CA}$ is the reorientation function using the charge arm vector; $C_{1,dih}$ is the intermittent dihedral autocorrelation for cis-trans interconversion. Adapted from Ref. [11] with permission from the PCCP Owner Societies.

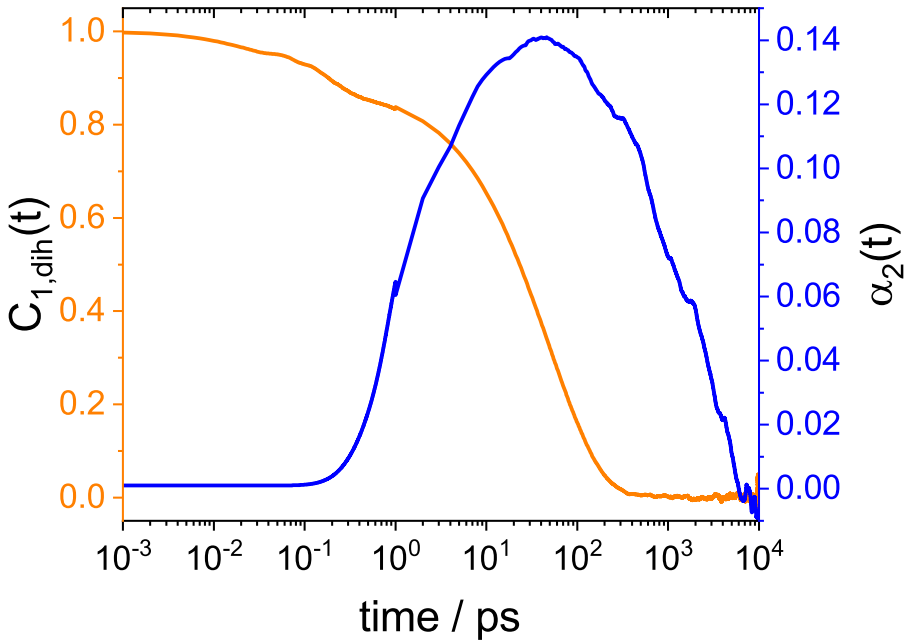

Figure 13: The α_2 parameter, and also the dihedral TCF again. Adapted from Ref. [11] with permission from the PCCP Owner Societies.

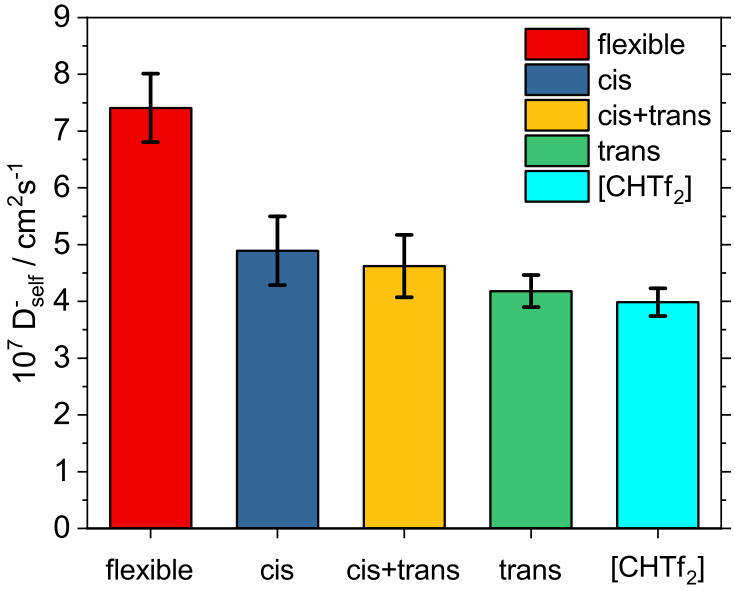

Figure 14: Anion diffusion obtained *via* the mean squared displacements. Adapted from Ref. [11] with permission from the PCCP Owner Societies.

6 Disclaimer

Everyone knows it, but it should be mentioned anyway:

TO ERR IS HUMAN.

I give no warranty for any results, or anything else for that matter. If you have a problem, drop me an email; I will give advice where feasible.

Best,
Frederik Philippi

This work was funded by the Imperial President's PhD Scholarship from October 2018 to June 2022.
This work is funded by the Postdoctoral Fellowships for Research in Japan of the Japan Society for the Promotion of Science.

References

[1] Julian B B Beckmann et al. "Molecular Dynamics of Ionic Liquids from Fast-Field Cycling NMR and Molecular Dynamics Simulations". In: *The Journal of Physical Chemistry B* 126.37 (Sept. 2022), pp. 7143–7158. ISSN: 1520-6106. DOI: 10.1021/acs.jpcb.2c01372. URL: https://pubs.acs.org/doi/10.1021/acs.jpcb.2c01372.

[2] M. Brehm et al. "TRAVIS—A free analyzer for trajectories from molecular simulation". In: *The Journal of Chemical Physics* 152.16 (Apr. 2020), p. 164105. ISSN: 0021-9606. DOI: 10.1063/5.0005078. URL: https://doi.org/10.1063/5.0005078%20http://aip.scitation.org/doi/10.1063/5.0005078.

[3] Martin Brehm and Barbara Kirchner. "TRAVIS - A Free Analyzer and Visualizer for Monte Carlo and Molecular Dynamics Trajectories". In: *Journal of Chemical Information and Modeling* 51.8 (Aug. 2011), pp. 2007–2023. ISSN: 1549-9596. DOI: 10.1021/ci200217w. URL: https://pubs.acs.org/doi/10.1021/ci200217w.

[4] G. Gradenigo et al. “Einstein Relation in Systems with Anomalous Diffusion”. In: *Acta Physica Polonica B* 44.5 (2013), p. 899. ISSN: 0587-4254. DOI: 10.5506/APhysPolB.44.899. URL: http://www.actaphys.uj.edu.pl/vol44/abs/v44p0899.

[5] Jean-Pierre Hansen and Ian R. McDonald. “Molecular Liquids”. In: *Theory of Simple Liquids*. Elsevier, 2013, pp. 455–510. DOI: 10.1016/B978-0-12-387032-2.00011-8. URL: https://linkinghub.elsevier.com/retrieve/pii/B9780123870322000118.

[6] Philipp Honegger et al. “Understanding the Nature of Nuclear Magnetic Resonance Relaxation by Means of Fast-Field-Cycling Relaxometry and Molecular Dynamics Simulations - The Validity of Relaxation Models”. In: *Journal of Physical Chemistry Letters* 11.6 (Mar. 2020), pp. 2165–2170. ISSN: 19487185. DOI: 10.1021/acs.jpclett.0c00087. URL: https://pubs.acs.org/doi/10.1021/acs.jpclett.0c00087.

[7] Hemant K. Kashyap et al. “How Is Charge Transport Different in Ionic Liquids and Electrolyte Solutions?” In: *The Journal of Physical Chemistry B* 115.45 (Nov. 2011), pp. 13212–13221. ISSN: 1520-6106. DOI: 10.1021/jp204182c. URL: http://pubs.acs.org/doi/abs/10.1021/jp204182c.

[8] Hualin Li et al. “The relationship between ionic structure and viscosity in room-temperature ionic liquids”. In: *The Journal of Chemical Physics* 129.12 (Sept. 2008), p. 124507. ISSN: 0021-9606. DOI: 10.1063/1.2978378. URL: http://aip.scitation.org/doi/10.1063/1.2978378.

[9] Jesse G. McDaniel and Arun Yethiraj. “Understanding the Properties of Ionic Liquids: Electrostatics, Structure Factors, and Their Sum Rules”. In: *The Journal of Physical Chemistry B* 123 (2019), pp. 3499–3512. ISSN: 1520-6106. DOI: 10.1021/acs.jpcb.9b00963.

[10] Frederik Philippi et al. “Charge transfer and polarisability in ionic liquids: a case study”. In: *Phys. Chem. Chem. Phys.* 24 (5 2022), pp. 3144–3162. DOI: 10.1039/D1CP04592J. URL: http://dx.doi.org/10.1039/D1CP04592J.

[11] Frederik Philippi et al. “Flexibility is the key to tuning the transport properties of fluorinated imide-based ionic liquids”. In: *Chemical Science* 13.32 (2022), pp. 9176–9190. ISSN: 2041-6520. DOI: 10.1039/D2SC03074H. URL: http://xlink.rsc.org/?DOI=D2SC03074H.

[12] Daniel Rauber et al. “Curled cation structures accelerate the dynamics of ionic liquids”. In: *Physical Chemistry Chemical Physics* 23.37 (2021), pp. 21042–21064. ISSN: 1463-9076. DOI: 10.1039/D1CP02889H. URL: http://pubs.rsc.org/en/Content/ArticleLanding/2021/CP/D1CP02889H%20http://xlink.rsc.org/?DOI=D1CP02889H.

[13] Chang Yun Son et al. “Proper Thermal Equilibration of Simulations with Drude Polarizable Models: Temperature-Grouped Dual-Nosé–Hoover Thermostat”. In: *The Journal of Physical Chemistry Letters* 10.23 (Dec. 2019), pp. 7523–7530. ISSN: 1948-7185. DOI: 10.1021/acs.jpclett.9b02983. URL: https://pubs.acs.org/doi/10.1021/acs.jpclett.9b02983.

[14] J. Trullàs and J. A. Padró. “Diffusion in multicomponent liquids: A new set of collective velocity correlation functions and diffusion coefficients”. In: *The Journal of Chemical Physics* 99.5 (Sept. 1993), pp. 3983–3989. ISSN: 0021-9606. DOI: 10.1063/1.466191. URL: http://aip.scitation.org/doi/10.1063/1.466191.

[15] J. Trullàs and J. A. Padró. “Self- and cross-velocity correlation functions and diffusion coefficients in liquids: A molecular dynamics study of binary mixtures of soft spheres”. In: *Physical Review E* 50.2 (Aug. 1994), pp. 1162–1170. ISSN: 1063-651X. DOI: 10.1103/PhysRevE.50.1162. URL: https://link.aps.org/doi/10.1103/PhysRevE.50.1162.

[16] Alexander Wulf et al. “Molecular reorientation in ionic liquids: A comparative dielectric and magnetic relaxation study”. In: *Chemical Physics Letters* 439.4-6 (May 2007), pp. 323–326. ISSN: 00092614. DOI: 10.1016/j.cplett.2007.03.084. URL: https://linkinghub.elsevier.com/retrieve/pii/S0009261407003922.