A New Perspective on the Order-n Algorithm for Computing Correlation Functions

David Dubbeldam, Denise Ford, Donald Ellis, Randall Snurr

Molecular Simulation
Journal of
Experimental Nanoscience

Taylor & Francis
Taylor & Francis Group

A New Perspective on the Order-n Algorithm for Computing Correlation Functions

Journal:	*Molecular Simulation/Journal of Experimental Nanoscience*
Manuscript ID:	GMOS-2008-0273.R1
Journal:	Molecular Simulation
Date Submitted by the Author:	05-Feb-2009
Complete List of Authors:	Dubbeldam, David; University of Amsterdam, Van 't Hoff Institute for Molecular Sciences Ford, Denise; Northwestern University, Chemical & Biological Engineering Department Ellis, Donald; Northwestern University, Department of Physics and Astronomy Snurr, Randall; Northwestern University, Chemical & Biological Engineering Department
Keywords:	correlations, diffusion, order-n

A New Perspective on the Order-n Algorithm for Computing Correlation Functions

David Dubbeldam[1], Denise C. Ford[1], Donald E. Ellis[2], and Randall Q. Snurr[1]

[1] *Chemical and Biological Engineering Department, Northwestern University, 2145 Sheridan Road, Evanston IL 60208 USA*

[2] *Department of Physics and Astronomy, Northwestern University, 2145 Sheridan Road, Evanston IL 60208 USA*

(Dated: February 5, 2009)

Abstract

A method to measure correlations is presented that can be shown to be identical to the original 'order-n algorithm' from Frenkel and Smit (Understanding Molecular Simulation, Academic Press, 2002). In contrast to their work, we present the algorithm without the use of 'block sums of velocities'. We show that the algorithm gives identical results compared to standard correlation methods for the time points at which the correlation is computed. We apply the algorithm to compute diffusion of methane and benzene in the metal-organic framework IRMOF-1 and focus on the computation of the mean-squared displacement, the velocity autocorrelation function, and the angular velocity autocorrelation function. Other correlation functions can readily be computed using the same algorithm. The savings in computer time and memory result from a reduction of the number of time points, as they can be chosen non-uniformly. In addition, the algorithm is significantly easier to implement than standard methods. Source code for the algorithm is given.

I. INTRODUCTION

Before working on special and general relativity, Albert Einstein published papers on diffusion, viscosity, and the photo-electric effect. Diffusion had been studied extensively by that time, starting with the pioneering work of Fick but was described in a completely phenomenological framework. Einstein proposed that Brownian motion of particles was basically the same process as diffusion. He connected the macroscopic process of diffusion with the microscopic thermal motion of individual molecules, proposing that $\langle x^2 \rangle = 6Dt$. This fundamental equation relates the average square of the molecular displacements $\langle x^2 \rangle$ to the self-diffusion coefficient D and the time t [1]. In the 1950s, Green and Kubo proved an exact expression relating linear transport coefficients (including the self-diffusivity) to integrals over time-correlation functions [2–5]. Green was the first to obtain expressions involving time-correlation function for coefficients of shear and bulk viscosity, thermal conductivity, diffusion and thermal diffusion. The expressions are based on the principle that the dynamical underlying process is a Markov process and that the deviations from thermal equilibrium are small. A rigorous general formalism of transport processes is presented by McQuarrie [6]. The formal equivalence of the Green-Kubo and Einstein expressions for self-and transport coefficients is well known. However, from a practical point of view, the Einstein formulation has the advantage that the integration over the velocities is already carried out at each time step by the integration scheme and does not need to be performed afterwards. This leads to less statistical errors and the interval between frames where data are stored to disk can be taken longer [7, 8]. Both formulations are frequently used in molecular dynamics computer simulations to compute self- and transport coefficients [8, 9].

Conventional methods to measure correlation functions are unable to measure fast and slow decay simultaneously. This limitation arises due to the fixed sampling frequency. A high sample frequency leads to large memory requirements as well as high cpu-time demands. With a low sampling frequency one can obtain long-time correlations, but any fast decay will be missed. The order-n algorithm by Frenkel and Smit allows for an adjustable sampling frequency, and fast and slow decay can be sampled simultaneously at minimal computational cost. The method as described in Ref. [10] is presented using block sums of velocities, blocked averaged velocities and coarse graining. In this paper, we revisit the algorithm and present versions for the Einstein and the Green-Kubo formulation. The reformulation of

the algorithm without the use of blocked averaged velocities is easier to understand in our opinion.

The remainder of this paper is organized as follows. After a short summary of the background material, we describe 3 algorithms in increasing efficiency and show that the last one is equal to the order-n algorithm of Frenkel and Smit, albeit that our approach omits the block sums of velocities in both the derivation and the implementation. In fact, this variant gives results in exact agreement with the conventional algorithm for the chosen time points. In the result section we show for several systems the advantages of the non-conventional approach over the conventional method for both the Einstein and Green-Kubo formalisms. The appendix contains a basic outline of the code.

II. BACKGROUND

We focus here on the correlation functions required to compute the self-diffusion coefficients in fluids or nanoporous materials. Other transport coefficients such as the transport diffusivities, bulk and shear viscosity, thermal conductivity etc. can be calculated in a similar fashion from different correlation functions [6]. The discussion presented here applies to these as well. For self-diffusion, two main routes are adapted: (a) the Einstein equation, relating the self-diffusivity and the mean-squared displacement (MSD), and (b) the Green-Kubo formulation, relating the self-diffusivity to the integral of the velocity autocorrelation function (VACF). The Green-Kubo and Einstein formalisms can be applied to self-diffusivity as well as to transport (or Fickian) diffusion. Transport diffusivity was originally described by Fick in a non-equilibrium diffusion framework. Even transport diffusivities can nowadays be computed from equilibrium simulations using the Einstein and Green-Kubo formalisms. Some recent work on self- and/or transport diffusion in nanoporous materials include Refs. [11–16].

In MD simulations [8, 10, 17], successive configurations of the system are generated by integrating Newton's laws of motion, which then yields a trajectory that describes the positions, velocities and accelerations of the particles as they vary with time. The self-diffusivity describes the motion of individual particles. In an equilibrium molecular dynamics simulation the self-diffusion coefficient D_α of component α is computed by taking the slope

of the mean-squared displacement at long times

$$D_\alpha = \frac{1}{2dN_\alpha} \lim_{t\to\infty} \frac{d}{d\,t} \left\langle \sum_{i=1}^{N_\alpha} \left(\mathbf{r}_i^\alpha(t) - \mathbf{r}_i^\alpha(0)\right)^2 \right\rangle \tag{1}$$

where N_α is the number of molecules of component α, d is the spatial dimension of the system, t is the time, and $\mathbf{r}_i^\alpha$ is the center-of-mass of molecule i of component α. Equivalently, D_α is given by the time integral of the velocity autocorrelation function

$$D_\alpha = \frac{1}{dN_\alpha} \int_0^\infty \left\langle \sum_{i=1}^{N_\alpha} \mathbf{v}_i^\alpha(t) \cdot \mathbf{v}_i^\alpha(0) \right\rangle dt \tag{2}$$

where $\mathbf{v}_i^\alpha$ is the center-of-mass velocity of molecule i of component α. Rotational diffusion can be studied using the angular velocity autocorrelation function

$$D_\alpha^R = \frac{1}{dN_\alpha} \int_0^\infty \left\langle \sum_{i=1}^{N_\alpha} \boldsymbol{\omega}_i^\alpha(t) \cdot \boldsymbol{\omega}_i^\alpha(0) \right\rangle dt \tag{3}$$

where $\boldsymbol{\omega}$ is the angular velocity. Eq. 1 is known as the Einstein equation and Eq. 2 is often referred to as the Green-Kubo relation. Similar equations exist for computing transport (collective) diffusion [15, 18, 19].

The Einstein and Green-Kubo equations given above can be applied to each x, y, z-direction individually (when the dimension of the system is taken in each case as $d = 1$), applied to the two dimensional case $d = 2$, or applied to the three dimensional system $d = 3$. In this case the directionally averaged diffusion coefficient is given by

$$D = \frac{D_x + D_y + D_z}{3} \tag{4}$$

III. ALGORITHMS FOR COMPUTING CORRELATIONS

A. Conventional algorithm

The conventional algorithm to measure autocorrelation functions can be implemented in several ways. Rapaport [17] presented the method as follows. Figure 1 shows the general framework for computing any kind of correlation function. In this example, time indices 80 to 95 are shown. First, we need a buffer to store the correlation function. The size of this buffer is chosen in advance and thus limits the correlation function to a predefined maximum time interval. At time index 80, an origin is stored. The data after the origin are

then correlated with the origin. In this simplified example the buffer is of size 10, so the maximum correlation time is 9 Δt, where $\Delta t = \delta t \times \tau$ is the integration time step δt times the sampling interval τ. The sampling interval τ is taken as 1 in Fig. 1 but it is often 5-10 integration steps in practice. After 10 time sampling steps, the buffer is full. In general, the time average of a property A in a simulation is computed as $\langle A \rangle = \frac{1}{N} \sum_{i=1}^{N} A_i$. Here, A_i is the current buffer that is full. We keep track of the summation of all these values during the simulation in an array denoted as the *accumulation buffer* (not shown in the figure) and a counter N keeps track of the number of buffers added. The average correlation function can be plotted at the end of the run or anytime during the run by printing the accumulated buffer divided by the counter. After the update of the accumulated buffer, a new time origin is stored, and the process is repeated. An important improvement is the use of overlapping buffers. In the example, not just one, but three buffers are used, each with a different offset in time. This offset is evenly spaced. Each time step contributes multiple data points to the various buffers and therefore improves the efficiency of the algorithm. Ideally, the overlap should be confined to time intervals over which the correlation between measurements has vanished, i.e. using 10 buffers in this example does not produce the *maximum* improvement in accuracy because successive samples of the buffers are usually correlated.

Each of the buffers contains a full sample of the autocorrelation function, which is added to the accumulated ACF when completely filled. An alternative is to store just the origins (instead of the buffers), and add the contribution to the accumulated ACF immediately while keeping track with an additional array of the amount of times a contribution has been added *per index*. This adds one array, but the need for storing the buffers is removed. This version is similar to the conventional algorithm present in Allen and Tildesley [8] and Frenkel and Smit[10].

B. Window algorithm

As mentioned, taking every step as a new time origin does not give the optimal improvement in efficiency, because the amount of cpu time increases without a corresponding gain in accuracy for highly correlated successive measurements. However, if the cpu time penalty is limited or the correlation between measurements is reduced, it could be advantageous to use every index as a new time origin. One can reduce the amount of correlation between

samples by changing the sample frequency from, for example, every time step to every 10 time steps. The increase in cpu time can be limited by using a single buffer containing the values to be correlated for a certain buffer length. At each time step the elements in the buffer can be correlated with the first element.

There are two basic ways of computing correlations: "post-processing" after the simulation has finished and "on-the-fly" during the simulation. Figure 2 shows a post-processing example by imagining all the values produced by the simulation as one long array containing positions for MSD, velocities for the VACF, etc. The ACF is computed by 'sliding' the ACF buffer over all the data to the right and taking a sample every step, or every few steps. The computed ACF sample is added to the accumulated buffer. The buffer can be viewed as a 'window' over the data produced by the simulation. A wider window provides a correlation function that is longer in time. The downside of post-processing is the extensive file input/output and storage requirements. However, one is always able to reprocess the data afterwards.

Figure 3 shows the "on-the-fly" alternative, here for the MSD. In this example, a block of data of size 10 is used to store the last 10 positions, from $r(-10\Delta t)$ for the left-most element corresponding to the position at time index 100 to $r(-\Delta t)$ corresponding to the position at time index 109 (both are relative to the current time index 110). The left-most element is the origin in time, and the other positions are relative to that value. From the block data one can easily compute the MSD and add it to the accumulated array. The resulting graph is shown in the top. The current time index is 110. For an update by Δt the data in the block are simply shifted to the left and the current value is copied to the most right element in the block data.

C. Multiple window algorithm

A downside of the window technique is that the buffer size has to be chosen in advance. A sample frequency at every time step leads to missing the long-time data (due to memory and cpu constraints), while a sample frequency of every 100th time step misses the first 100.

We propose a new method in the same spirit as the conventional method. The key idea is to use several windows using the conventional window technique, but each of the windows has a different sampling frequency [10]. In Figure 4 we show an example of three buffers of

size 10. The first buffer (block 0) samples every step, the second buffer (block 1) samples every 10 steps, the third buffer (block 2) every 100 steps, and buffer n samples every 10^n time steps. At every step, we examine whether to update the buffers. If the time step is a multiple of the buffer size to the power n we sample a value for the buffer n. The whole correlation function can be constructed by placing the accumulated buffers next to each other, from left to right, each producing a different time region for the complete graph (see MSD results in the next section).

D. Comparison to the order-n algorithm

In our approach we have simply chosen the position (instead of the sum of the velocities) to compute the mean-squared displacement. For this algorithm, the value inside the buffers are shifted to left when updating the right-most element. Note that when using positions, one needs to be careful to account properly for periodic boundary effects. Instead of the position, one can also use the velocity, even for the mean-squared displacement, if one replaces shifting to the left by *adding* the value in a given element to the value in the element on the left. This will lead to a summation of velocities which can be related to the position by $\Delta r = v \times \Delta t$. An integration algorithm like velocity Verlet exactly obeys this relation. The version using summed velocities is equivalent to the order-n algorithm presented in Frenkel and Smit. The order-n algorithm is depicted in Figure 5. The order-n algorithm samples from the left-most elements of the next lower block, except for the first block which samples from the velocities. This is not essential, because the current value has the same time separation as the left-most elements and one can just as well always sample from the current value, provided one uses the position and not the sum of the velocities. Therefore our code is shorter and perhaps easier to understand, because it is basically the conventional window approach extended to multiple windows that each sample at a different frequency. Our presentation of the algorithm allows us to make three additional statements about the approach:

- The algorithm gives equal results to the conventional algorithm for all time points chosen. The main advantage is the reduction of the number of time points and control of sample frequency.

- Block sums of velocities are not essential. The use of positions instead of velocities is more convenient for mean-squared displacements.

- The order-n method for the MSD correlates backwards in time. For time correlations that are time-reversible, this presents no problem for systems at equilibrium. The multiple window technique can be simplified even more for this case (see the code in the appendix).

As mentioned by Frenkel and Smit, the required storage of data using this algorithm is the 'block-size' times 'the number of blocks', compared to the 'block-size' to the power 'number of blocks' for the conventional algorithm (for the same correlation range). The sample frequencies chosen in Ref. [10] lead to an order-n algorithm. The gain originates from a significant reduction of the number of time points. One is, of course, free to tune the number and spacing of sampling points to specific applications. We have therefore avoided the use of the term order-n. The number of floating point operations scales as t^2 for the conventional scheme, where t is the simulation time, and as t for the 'multiple-window' technique. Another often used method makes use of the Fourier technique, which reduces the conventional t^2 to $t \ln t$. However, this analysis assumes one would correlate longer for a longer simulation time. In practice, for the conventional algorithm one usually fixes the maximum correlation time in advance. No such restriction is necessary for the multiple windows technique, because new blocks can be allocated on the fly. Choosing a single block reduces the method to the conventional 'window' technique.

IV. RESULTS

We present here some correlation functions that are often used in the study of diffusion in nanoporous materials. The structure we use here is the metal-organic framework IRMOF-1 [20–24]. Methane is chosen as an adsorbate for its simplicity and we also study benzene as a rigid molecule to compute rotational diffusion.

Figure 6 shows the mean-squared displacement of methane at 298 K in IRMOF-1. IRMOF-1 is a prototypical metal-organic framework [20, 21]. The time step was 0.5 fs and the classical force field was taken from Ref. [25]. The framework was kept rigid. The ensemble was NVT using the Nose-Hoover chain method of Martyna and Tuckerman [26–

28]. The open symbols are the multiple window technique data, the lines are results for the conventional algorithm. The conventional algorithm uses a buffer size of 10000 and a sampling frequency of every step, every 100 steps, and every 1000 steps for the top, middle, and bottom graphs respectively. The multiple windows technique has 25 elements per block and 6 blocks. The different blocks are clearly distinguished in the log-log curves, because within a block the data has the same spacing in time, but each block corresponds to a different order of magnitude in time. The multiple window technique and the conventional algorithm give identical results for the chosen time points. However, the multiple window technique can capture both short and long times during a single run. The conventional algorithm is limited in time mainly because of memory storage. Figure 7 shows for the same system the velocity autocorrelation function using the conventional method and the multiple window technique. Again, the latter technique allows both short and long times to be measured. The integral of the VACF can be used to compute the diffusion coefficient.

A second case study is benzene in IRMOF-1. Benzene is simulated as a rigid molecule using a quaternion integration scheme [27]. Figure 8 shows that for runs that last 5 nanoseconds using a time step of 2.0 fs, no observable energy drift occurs. During the run the angular velocity autocorrelation was measured, see Figure 9. Several options are available: (a) reorientation of the molecule as a whole, (b) reorientation of the individual molecular axes, (c) the angular velocity in the laboratory framework, and (d) the angular velocity in the molecular frame [29]. Here, we used the last method, which allows the calculation of rotational diffusion coefficients along individual molecular axes. The rotational self-diffusivity is related to the area underneath the correlation function. As expected, the angular velocity-autocorrelation function in the molecular frame shows that rotation around the out-of-plane (x) axis is the fastest, whereas rotation around the short in-plane axes (y and z) is the slowest. The latter motion is severely restricted by the framework as evidenced by the reversal of sign. Compared to 10 benzene molecules per unit cell, the rotational diffusion is slower at 40 molecules per unit cell.

The main drawback of the Green-Kubo formulation is that, in principle, the integration limits of the autocorrelation function are from zero to infinity. Fortunately, correlation functions usually decay fast enough to allow a finite integration region. However, it remains difficult to distinguish noise from a real contribution to the diffusion coefficient, i.e. the long-time tail. The mean-squared displacement as plotted in Fig. 10 provides more prac-

tical guidelines how to accurately obtain diffusivities. At very short time scales the MSD has a quadratic dependence on time (a slope of two on a log-log plot). This is known as the ballistic regime, where particles on average do not yet collide. In nanoporous materials, an intermediate regime starts when particles are colliding with the framework and the other particles in the same confinement. Only when particles are able to escape the local environment and explore the full periodic lattice length l is the diffusional regime reached. The time required to travel an effective length l is related to the 'residence time'. It is the time a particle on average spends in a cage-like environment before escaping to the next repeating part of the pore space.

In the diffusive regime, the mean-squared displacement has bent over to attain a different slope and becomes linear with time (a slope of unity on a log-log plot). It is the long-time diffusion coefficient that is of interest for macroscopic diffusion. The start of the linear regime is often directly related to the squared length of the smallest repeating length, i.e. the unit cell length or the length of a cage. The region to use for fitting should preferably not include all data because data have a progressively bigger error bar as the correlation time increases. Note that for simulation length of T, one can have $T-1$ samples of Δt, but only 1 of length T. Correlation functions are therefore most accurate at small times and generally decrease greatly in accuracy for longer times. In Fig. 10 it is clear that the data at the longest times are unreliable, because they are not linear. Moreover, the MSD data at 450 K and 500 K cross. It is therefore advisable to restrict the fitting to a smaller region starting from where the linear regime starts.

From a practical point of view the error can be reduced even further by a "parallel farming approach". As an example, instead of running a single job, one can run several jobs at different temperatures. Very often, when plotting $\ln(D)$ vs $1/T$ the data shows a straight line (Arrhenius type behavior). Another approach is to run several jobs as a function of loading around the region of interest. The essential point is that from physical arguments one can reason that the curves should be rather smooth and continuous. Thus the accuracy of each of the individual runs can be examined conveniently from a broader set of simulation data. In one possible approach one could simulate without a set end time, examine the MSD's once in while, study the Arrhenius behavior, and decide to stop the simulations when a sufficient accuracy has been reached. As an example, Figure 10 shows the behavior of benzene in IRMOF-1 as a function of temperature. After 20 nanoseconds, the

diffusivities have converged sufficiently to achieve Arrhenius behavior. Note that, of course, there are many systems that show non-Arrhenius behavior [30], but also these systems usually show different regimes that by themselves are linear. It is always beneficial to view sets of simulation data in a bigger picture to estimate the accuracy, rather then focusing on individual runs.

V. CONCLUSIONS

We have presented a method to sample correlation functions that is able to capture short and long times simultaneously. As an example, the diffusion of methane and benzene in the nanoporous material IRMOF-1 was studied. The algorithm gives identical results compared to the conventional algorithm, but at minimal computational cost. The summed velocity variant of the algorithm is almost identical to the Frenkel and Smit order-n algorithm. However, the presented variant using positions for the mean-squared displacement is more convenient. The implementation is straightforward for any correlation function and perhaps even simpler than the conventional algorithms.

Acknowledgments

This material is based upon work supported by the National Science Foundation under the following NSF programs: Partnerships for Advanced Computational Infrastructure, Distributed Terascale Facility (DTF) and Terascale Extensions: Enhancements to the Extensible Terascale Facility. This work was also supported by the National Science Foundation (CTS-0507013) and the Defense Threat Reduction Agency. Correspondence should be addressed to Randall Q. Snurr (email: snurr@northwestern.edu).

Appendix

We present two general c-routines for computing correlation functions, one using the Einstein formulation and one for the Green-Kubo formalism. Both routines should be called using the argument 'ALLOCATE' before the MD run, to allocate the required memory (can be dynamically or using static arrays); using 'SAMPLE' during the MD run to sample during

the run; using 'PRINT' to output the data to files; and optionally using 'DEALLOCATE' to free the memory (the operating system will do that anyway when the program finishes). These names are defined as an enumeration in c:

```
enum{ALLOCATE,INITIALIZE,SAMPLE,PRINT,FINALIZE};
```

Many unneeded lines are removed for clarity, such as error checking etc. A more detailed implementation would include diffusion in x, y, z directions independently, computation of collective diffusion, etc. Note that in c, arrays start from index 0 (in contrast to Fortran where arrays start at index 1). The characters '//' denote comments and the '%' operator means *modulus*. We note that in a detailed implementation the 'fmod' operator for modulus on floating point numbers is available to avoid overflow when using integers. The routines 'GetCenterOfMassPosition(int m)' and 'GetCenterOfMassVelocity(int m)' return the center-of-mass position and center-of-mass velocity for molecule m, respectively. They return a 'VECTOR' which is a structure with three elements 'x', 'y', and 'z'.

```
typedef struct vector
{
  double x;
  double y;
  double z;
} VECTOR;
```

For simplicity we use here static allocation using global variables:

```
#define MAX_NUMBER_OF_BLOCKS 50
#define MAX_NUMBER_OF_BLOCKELEMENTS 25
#define MAX_NUMBER_OF_MOLECULES 1000

int BlockLength[MAX_NUMBER_OF_BLOCKS];
VECTOR BlockData[MAX_NUMBER_OF_BLOCKS][MAX_NUMBER_OF_MOLECULES][MAX_NUMBER_OF_BLOCKELEMENTS];
double MsdCount[MAX_NUMBER_OF_BLOCKS][MAX_NUMBER_OF_BLOCKELEMENTS];
double MsdAv[MAX_NUMBER_OF_BLOCKS][MAX_NUMBER_OF_BLOCKELEMENTS];
double VacfCount[MAX_NUMBER_OF_BLOCKS][MAX_NUMBER_OF_BLOCKELEMENTS];
```

```
double VacfAv[MAX_NUMBER_OF_BLOCKS][MAX_NUMBER_OF_BLOCKELEMENTS];
```

The Einstein routine has the following general outline:

```
int SampleMeanSquareDisplacementMultipleWindows(int Switch)
{
  int i,j,k,index,index_origin;
  int CurrentBlock,CurrentBlocklength;
  VECTOR value,drift,origin;
  FILE *FilePtr;
  char buffer[256];

  switch(Switch)
  {
    case ALLOCATE:
      // allocate memory
      break;
    case SAMPLE:
      // determine current number of blocks
      NumberOfBlocks=1;
      i=count/NumberOfBlockElements;
      while(i!=0)
      {
        NumberOfBlocks++;
        i/=NumberOfBlockElements;
      }

      // loop over all the blocks to test which blocks need sampling
      for(CurrentBlock=0;CurrentBlock<NumberOfBlocks;CurrentBlock++)
      {
        // test for blocking operation, i.e. when count is a multiple
        // of NumberOfBlockElements^CurrentBlock
```

```
if ((count)%((int)pow(NumberOfBlockElements,CurrentBlock))==0)
{
  // increase the current block-length
  BlockLength[CurrentBlock]++;

  // compute the current length of the block, limited to size 'NumberOfBlockElements'
  CurrentBlocklength=MIN(BlockLength[CurrentBlock],NumberOfBlockElements);

  // loop over the molecules in the system
  for(k=0;k<NumberOfMolecules;k++)
  {
    // shift to the left, set last index to the correlation value
    for(i=1;i<NumberOfBlockElements;i++)
      BlockData[CurrentBlock][k][i-1]=BlockData[CurrentBlock][k][i];
    BlockData[CurrentBlock][k][NumberOfBlockElements-1]=GetCenterOfMassPosition(k);

    // get the origin, take into account that blocks can be partially filled
    index_origin=NumberOfBlockElements-CurrentBlocklength;
    origin=BlockData[CurrentBlock][i][index_origin];

    // sample msd using proper reference position
    for(i=0;i<CurrentBlocklength;i++)
    {
      MsdCount[CurrentBlock][i]+=1.0;
      MsdAv[CurrentBlock][i]+=
        SQR(BlockData[CurrentBlock][k][index_origin+i].x-origin.x)+
        SQR(BlockData[CurrentBlock][k][index_origin+i].y-origin.y)+
        SQR(BlockData[CurrentBlock][k][index_origin+i].z-origin.z);
    }
  }
}
```

```
      }
      // count the current sampling
      count++;
      break;
    case PRINT:
      FilePtr=fopen("output_msd.dat","w");

      for(CurrentBlock=0;CurrentBlock<MIN(MaxNumberOfBlocks,NumberOfBlocks);CurrentBlock++)
      {
        CurrentBlocklength=MIN(BlockLength[CurrentBlock],NumberOfBlockElements);
        for(j=1;j<CurrentBlockLength;j++)
        {
          // write time-index
          fprintf(FilePtr,"%g ",(double)(j*DeltaT*pow(NumberOfBlockElements,CurrentBlock)));

          // isotropic self-diffusion
          if(MsdCount[CurrentBlock][j]>0.0)
            fprintf(FilePtr,"%g\n",(double)(MsdAv[CurrentBlock][j]/MsdCount[CurrentBlock][j]));
        }
      }

      fclose(FilePtr);
      break;
    case DEALLOCATE:
      // free memory
      break;
}
```

The Green-Kubo routine, here for the velocity autocorrelation function, has the following general outline:

```
int SampleVelocityAutocorrelationFunctionMultipleWindows(int Switch)
```

```
{
  int i,j,k,index,index_origin;
  int CurrentBlock,CurrentBlocklength;
  VECTOR value,drift;
  FILE *FilePtr;
  char buffer[256];

  switch(Switch)
  {
    case ALLOCATE:
      // allocate memory
      break;
    case SAMPLE:
      // determine current number of blocks
      NumberOfBlocks=1;
      i=count/NumberOfBlockElements;
      while(i!=0)
      {
        NumberOfBlocks++;
        i/=NumberOfBlockElements;
      }

      // loop over all the blocks to test which blocks need sampling
      for(CurrentBlock=0;CurrentBlock<NumberOfBlocks;CurrentBlock++)
      {
        // test for blocking operation, i.e. when count is a multiple
        // of NumberOfBlockElements^CurrentBlock
        if ((count)%((int)pow(NumberOfBlockElements,CurrentBlock))==0)
        {
          // increase the current block-length
          BlockLength[CurrentBlock]++;
```

```
      // compute the current length of the block, limited to size 'NumberOfBlockElements'
      CurrentBlocklength=MIN(BlockLength[CurrentBlock],NumberOfBlockElements);

      // loop over the molecules in the system
      for(k=0;k<NumberOfMolecules;k++)
      {
        // shift to the left, set last index to the correlation value
        for(i=1;i<NumberOfBlockElements;i++)
          BlockData[CurrentBlock][k][i-1]=BlockData[CurrentBlock][k][i];
        BlockData[CurrentBlock][k][NumberOfBlockElements-1]=GetAdsorbateCenterOfMassVelocity(k);

        // get the origin, take into account that blocks can be partially filled
        index_origin=NumberOfBlockElements-CurrentBlocklength;
        origin=BlockData[CurrentBlock][i][index_origin];

        // sample vacf using proper reference velocity
        for(i=0;i<CurrentBlocklength;i++)
        {
          VacfCount[CurrentBlock][i]+=1.0;
          VacfAv[CurrentBlock][i]+=
            (BlockData[CurrentBlock][k][index_origin+i].x*origin.x)+
            (BlockData[CurrentBlock][k][index_origin+i].y*origin.y)+
            (BlockData[CurrentBlock][k][index_origin+i].z*origin.z);
        }
      }
    }
  }
  // count the current sampling
  count++;
  break;
```

```
    case PRINT:
      FilePtr=fopen("output_vacf.dat","w");

      for(CurrentBlock=0;CurrentBlock<MIN(MaxNumberOfBlocks,NumberOfBlocks);CurrentBlock++)
      {
        CurrentBlocklength=MIN(BlockLength[CurrentBlock],NumberOfBlockElements);
        for(j=1;j<CurrentBlocklength;j++)
        {
          if(VacfCount[CurrentBlock][j]>0.0)
          {
            fprintf(FilePtr,"%g %g\n",
                  (j*SampleEvery*DeltaT*pow(NumberOfBlockElements,CurrentBlock)),
                  (VacfAv[CurrentBlock][j]/VacfCount[CurrentBlock][j]),
          }
        }
      }

      fclose(FilePtr);
      break;
    case DEALLOCATE:
      // free memory
      break;
}
```

Lastly, we note that if the correlation function obeys time-reversibility, then the algorithm can be simplified. For example, the VACF becomes (only the difference is shown below):

```
for(k=0;k<NumberOfMolecules;k++)
{
  orgin=GetAdsorbateCenterOfMassVelocity(k);

  // shift to the right, set index 0 to the correlation value
```

```
    for(i=CurrentBlocklength-1;i>0;i--)
      BlockData[CurrentBlock][k][i]=BlockData[CurrentBlock][k][i-1];
    BlockData[CurrentBlock][k][0]=origin;

    // sample vacf using proper reference velocity
    for(i=0;i<CurrentBlocklength;i++)
    {
      VacfCount[CurrentBlock][i]+=1.0;
      VacfAv[CurrentBlock][i]+=(BlockData[CurrentBlock][k][i].x*origin.x)+
                               (BlockData[CurrentBlock][k][i].y*origin.y)+
                               (BlockData[CurrentBlock][k][i].z*origin.z);
    }
  }
```

This routine actually stores the data from left to right in the arrays using right-shifts and therefore correlates backwards in time when the current value is used as the time origin. Newton's equation of motion are time-reversible. Modern integrators are time-reversible [26] and even sympletic [27].

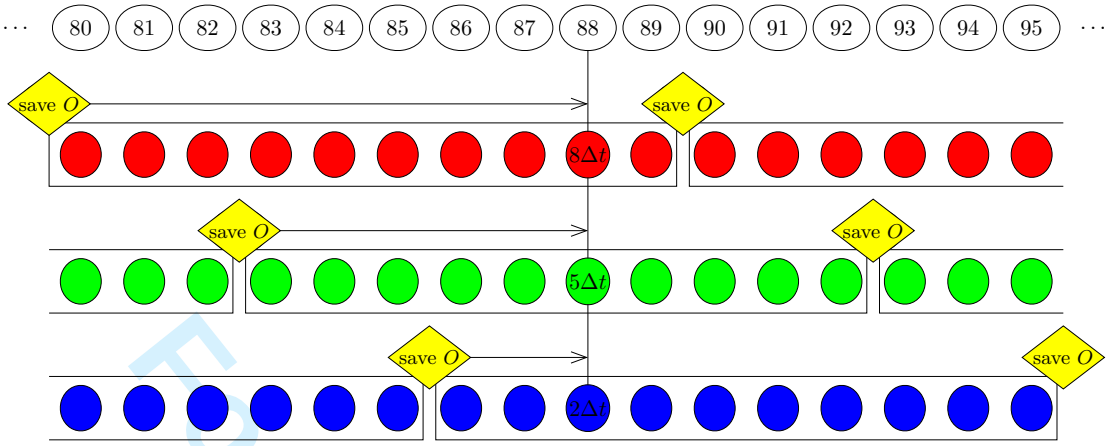

Figure 1: Conventional technique to sample correlation functions. Shown here are simulation steps 80 through 95 and 3 buffers of size 10 with different origins in time that are in simultaneous use. The time origins of the blocks are evenly spaced. The current simulation step is 88. At this step, the current value is combined with the stored origin of buffer 1 to compute the correlation at a time difference of $8\Delta t$. But it is also combined with the two other blocks for time differences of $5\Delta t$ and $2\Delta t$, respectively. The use of multiple buffers increases efficiency. Each of the buffers contains, when full, a sample of the ACF. At the end of the next time step 89, buffer 1 is full. The ACF is added to an array containing the accumulated ACFs (not shown in the figure), and the value at step 90 is stored as the new origin O. In real applications, about 10-20 buffers of size 100-500 are usually used, and often the sampling is only every 5-10 MD steps.

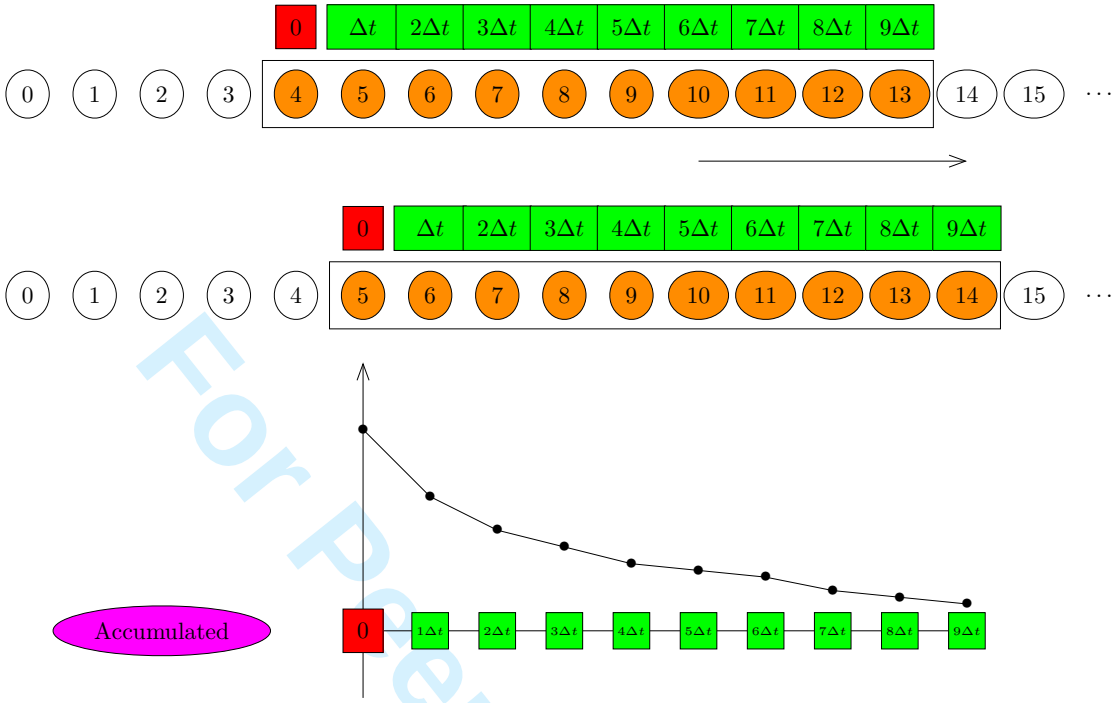

Figure 2: Window technique ("post-processing"): a single buffer is used which represents a window of values over time. In the example the buffer is of size 10, and therefore the correlation can be computed up to $9\Delta t$. Using a post-processing point of view where all the data is available after a simulation run, one can imagine moving the window from left to right over the simulation results and to take at every step a sample of the correlation function, i.e. each index in the (red+green) window is correlated with the (red) time origin. At each update the sample ACF is added to an array containing the accumulated ACF. The accumulated ACF divided by the amount of samples is the current *average* ACF. The resulting curve looks schematically like the bottom graph, a decaying function in time as properties become decorrelated due to the particle interactions.

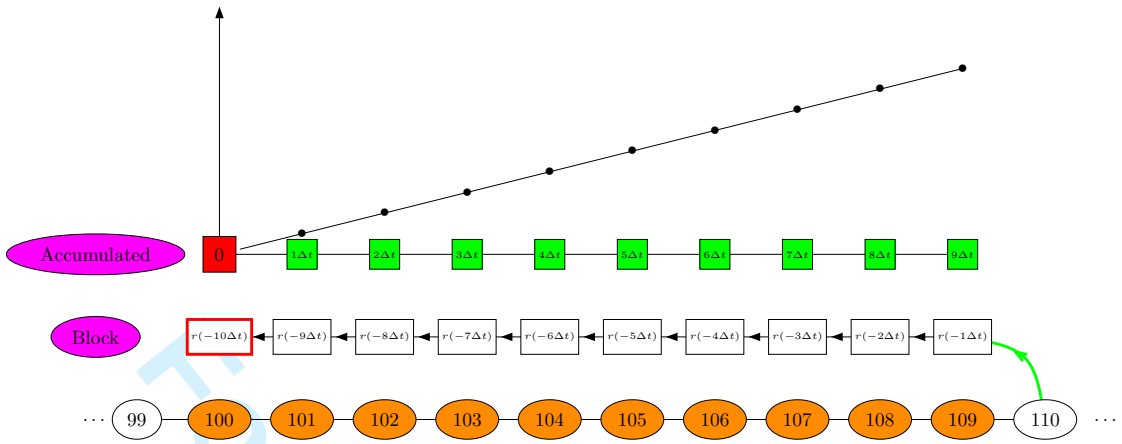

Figure 3: Window technique ("on-the-fly"): a single buffer is used which represents a window of values over time. The example is for computing the MSD and therefore positions are stored. For the VACF the velocities would be used. The buffer is of size 10 and contains the positions separated in time. The (red) left element is first in time ($r(-10\Delta t)$ with respect to the position at time index 110), and therefore serves as the origin. The elements of the block of data are correlated to the origin to compute the mean-squared displacement that is immediately added to an array containing the accumulated MSD. The accumulated MSD divided by the amount of samples is the current *average* MSD. The resulting graph is shown schematically at the top. In this MSD example, the current time step is 110 and the block-data actually contains the positions at time steps 100 up to and including 109. We update the block data by Δt by shifting the values to the left and placing the position of time step 110 in the right-most element (green line).

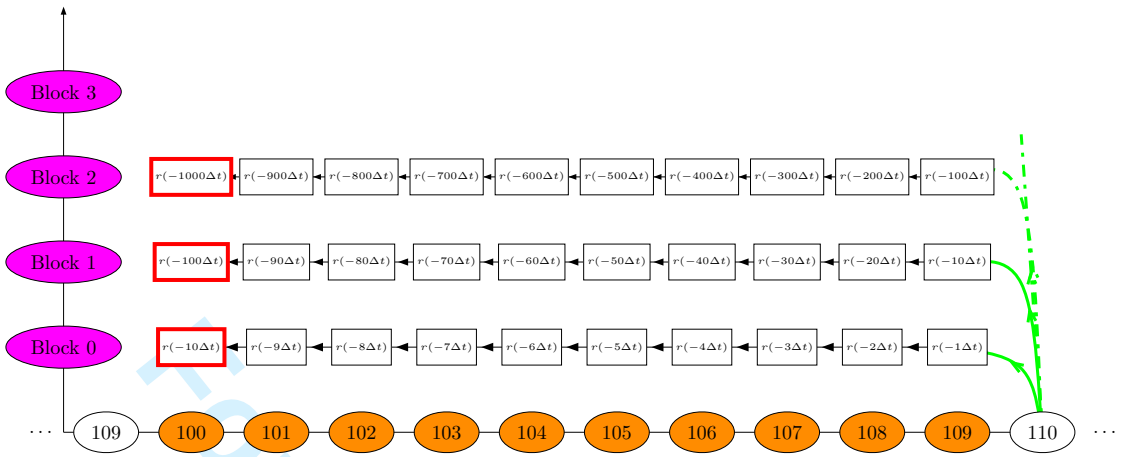

Figure 4: The multiple windows technique ("on-the-fly"). In this MSD example the block length is chosen as 10 and only the first 3 blocks are shown. Each block represents measurements at different time scales. Block 0 is sampled every Δt: the block is shifted to the left to update Δt in time and the new, last element is taken as the position of the particle. Block 1 samples every $10\Delta t$, Block 2 samples every $100\Delta t$. One can use the modulus operation to decide whether to update a block. Suppose we are at time index 110, then blocks 0 and 1 are updated, because 110 modulus 1 and 110 modulus 10 are zero. At every processing step, these blocks are used to update the appropriate parts of the MSD using the left (red) elements as the origins for the blocks. Each block element i stores the position with a time difference of $i \times \Delta t \times 10^{\text{Block}}$ compared to the current position of the particle. The position at index i and index 0 (the origin) are used to compute the MSD for that time interval and added to the accumulated MSD.

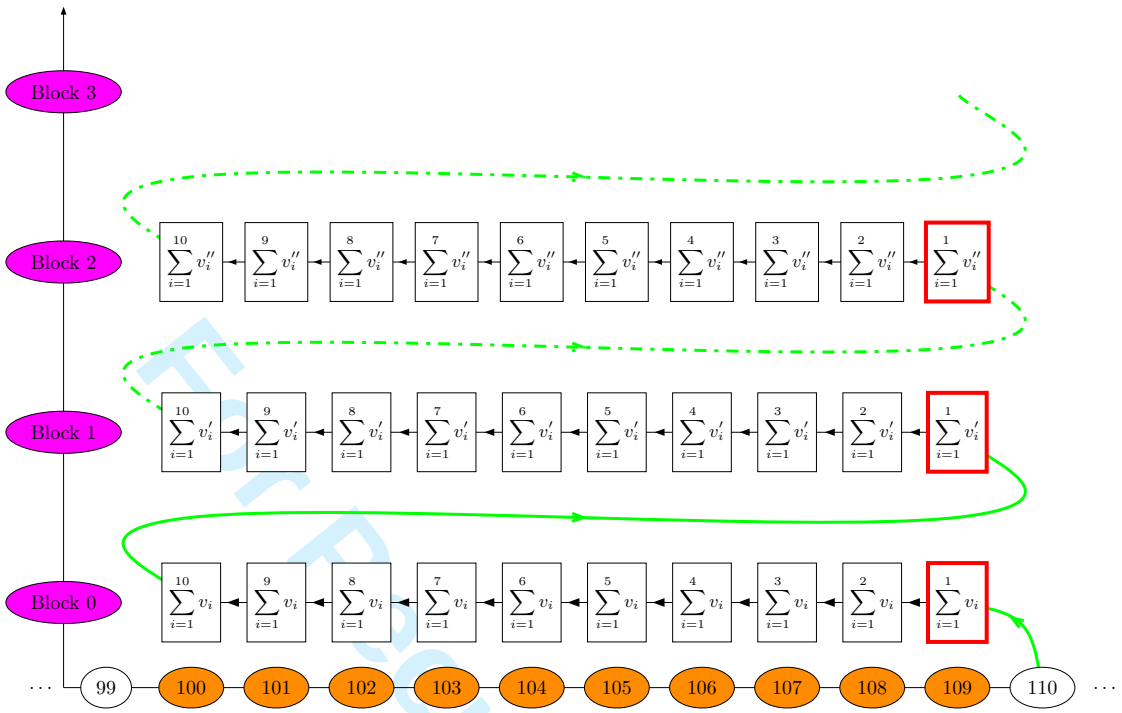

Figure 5: The order-n algorithm for the MSD from Ref. [10]. The difference with the multiple window technique is that instead of sampling from the current value at time step 110, the values of the left-most element of the next lower block are used. The current value is only used for the first block. A second difference is that velocities instead of positions are used. The update therefore consists of a left shift where the values are not overwritten, but *added* to the next left element. This is because a position difference is the *sum* of the previous velocities ($\Delta r = \Delta t \times \sum v$). A third difference is that the correlation is actually backwards in time (because the (red) origins are the last elements on the right which are the most recent values).

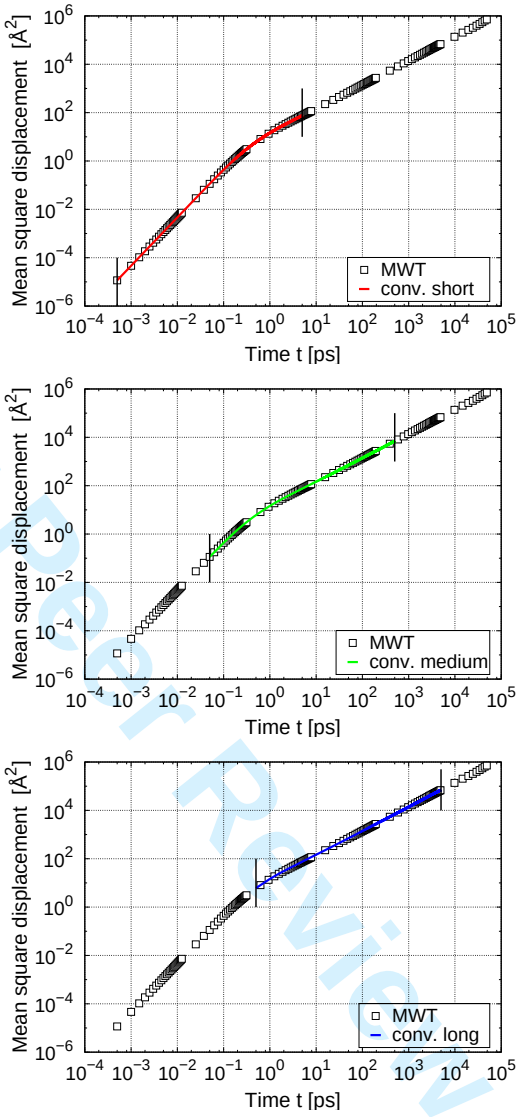

Figure 6: Mean-squared displacement of methane in IRMOF-1. The loading is 64 molecules per unit cell and the temperature is 298 K. The multiple window technique (MWT) is compared to the conventional algorithm using a buffer of size 10000 and three different sample frequencies: (top) every 10 integrations steps, (middle) every 100 steps, and (bottom) every 1000 steps. The integration time step was 0.5 fs.

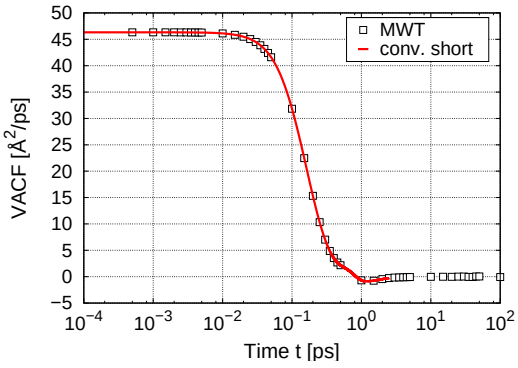

Figure 7: Velocity autocorrelation function of 64 methane molecules in a single unit cell of IRMOF-1 at 298K. The open symbols are data of the multiple window technique, the line is the data of the conventional algorithm using a buffer of size 10000 and sampling every 10 integration steps.

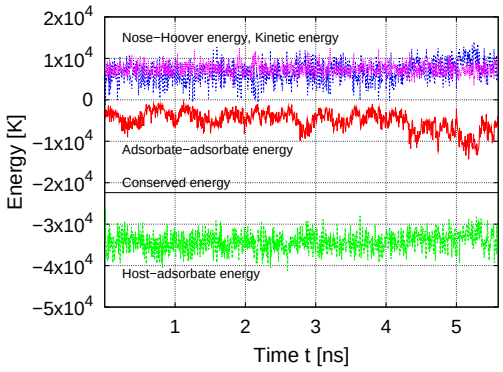

Figure 8: The energies and conserved quantity of 10 benzene molecules in a single unit cell of IRMOF-1 at 298 K during a MD run of 5 nanoseconds. Relative energy conservation is on the order of 10^{-4}.

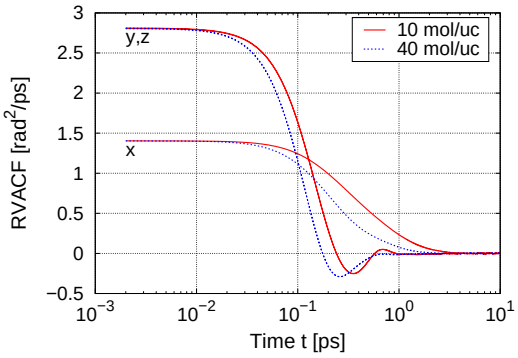

Figure 9: The angular velocity autocorrelation function of benzene, computed using the MWT technique, at 298 K and a loading of 10 molecules and 40 molecules per unit cell, respectively. The data for y and z coincide due to geometric symmetry.

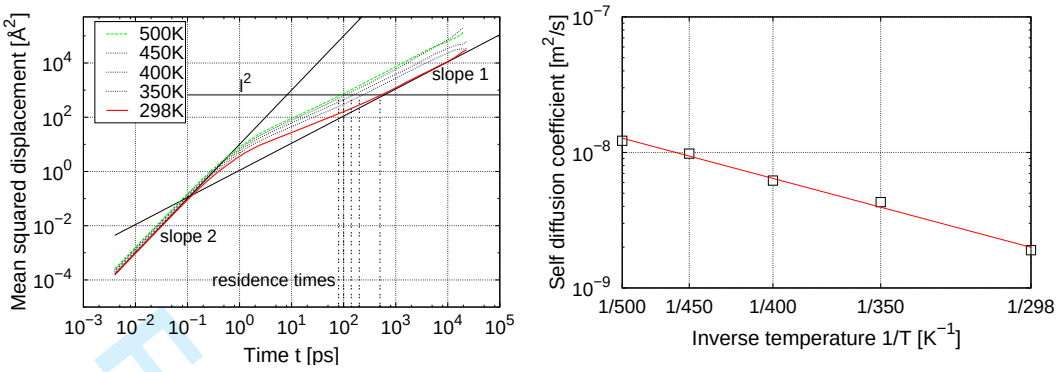

Figure 10: Diffusion of benzene in IRMOF-1 at 10 molecules per unit cell as a function of temperature: (left) the mean-squared displacements from top to bottom at 500, 450, 400, 350 and 298 K, (right) the Arrhenius behavior after 20 nanoseconds of simulation. The mean-squared displacements become linear after approximately l^2, where $l = 25.83$ Åis the periodic length of the IRMOF-1 unit cell. The intersections of the mean-squared displacements with l^2 are directly related to the residence times of the particle inside the cages. An Arrhenius plot after several nanoseconds shows scatter (not shown), implying the simulations are not properly converged yet. For this system, it takes simulations of at least 20 nanoseconds to obtain reliable results.

[1] Kärger, J., *Leipzig, Einstein, Diffusion* Leipziger Universitätsverlag; Leipzig, 2007.

[2] Green, M.S. *J. Chem. Phys.* **1952**, *20*, 1281-95.

[3] Green, M.S. *J. Chem. Phys.* **1954**, *22*, 398-413.

[4] Kubo, R. *J. Phys. Soc. Japan* **1957**, *12*, 570-586.

[5] Kubo, R.; Yokota, M.; Nakajima, S. *J. Phys. Soc. Japan* **1957**, *12*, 1203-1211.

[6] McQuarrie, D.A., *Statistical Thermodynamics* HarperCollinsPublishers Inc.; New York, 1968.

[7] Tepper, H. L.; Briels, W. J. *J. Chem. Phys.* **2002**, *116*, 9464-9474.

[8] Allen, M. P.; Tildesley, D. J., *Computer Simulation of Liquids* Clarendon Press; Oxford, 1987.

[9] Evans, D.J.; Morriss, D.G, *Statistical Mechanics of Nonequilibrium Liquids* Academic Press; London, 1990.

[10] Frenkel, D.; Smit, B., *Understanding Molecular Simulation 2nd edition* Academic Press; London, UK, 2002.

[11] Skoulidas, A.I.; Sholl, D. S. *J. Phys. Chem. A* **2003**, *107*, 10132-10141.

[12] Beerdsen, E.; Dubbeldam, D.; Smit, B. *J. Phys. Chem. B* **2006**, *110*, 22754-22772.

[13] Skoulidas, A. I.; Sholl, D. S. *J. Phys. Chem. B* **2005**, *109*, 15760-15768.

[14] Krishna, R.; van Baten, J.M. *Chem. Eng. Sci.* **2008**, *63*, 3120-3140.

[15] Dubbeldam, D.; Snurr, R.Q. *Mol. Sim.* **2007**, *33*, 305-325.

[16] Sarkisov, L.; Düren, T.; Snurr, R. Q. *Mol. Phys.* **2004**, *102*, 211-221.

[17] Rapaport, D. C., *The Art of Molecular Dynamics Simulation 2nd edition* Cambridge University Press; Cambridge, 2004.

[18] Theodorou, D. N.; Snurr, R. Q.; Bell, A. T., in *Comprehensive Supramolecular Chemistry*, edited by Alberti, G.; Bein, T. Pergamon Oxford; Oxford, 1996, Vol. 7, Chap. Chap. 18, pp. 507–548.

[19] Krishna, R.; van Baten, J. M. *J. Phys. Chem. B* **2005**, *109*, 6386-6396.

[20] Li, H.; Eddaoudi, M.; O'Keeffe, M.; Yaghi, O. M. *Nature* **1999**, *402*, 276-279.

[21] Eddaoudi, M.; Kim, J.; Rosi, N.; Vodak, D.; Wachter, J.; O'Keefe, M.; Yaghi, O. M. *Science* **2002**, *295*, 469-472.

[22] Yaghi, O. M.; O'Keeffe, M.; Ockwig, N. W.; Chae, H. K.; Eddaoudi, M.; Kim, J. *Nature* **2003**, *423*, 705-714.

[23] Kitagawa, S.; Kitaura, R.; Noro, S. *Nature* **2003**, *423*, 705-714.

[24] Snurr, R. Q.; Hupp, J. T.; Nguyen, S. T. *AIChE Journal* **2004**, *50*, 1090-1095.

[25] Dubbeldam, D.; Walton, K.S.; Ellis, D.E.; Snurr, R.Q. *Angew. Chem. Int. Ed.* **2007**, *46*, 4496-4499.

[26] Martyna, G. J.; Tuckerman, M.; Tobias, D. J.; Klein, M. L. *Mol. Phys.* **1996**, *87*, 1117-1157.

[27] Miller, T. F.; Eleftheriou, M.; Pattnaik, P.; Ndirango, A.; Newns, D.; Martyna, G. J. *J. Chem. Phys.* **2002**, *116*, 8649-8659.

[28] Tuckerman, M.E.; Alejandre, J.; Lopez-Rendon, R.; Jochim, A.L.; Martyna, G.J. *J. Phys. A* **2006**, *39*, 5629-5651.

[29] Jas, G.S.; Larson, E.J.; Johnson, C.K.; Kuczera, K. *J. Phys. Chem. A* **2000**, *104*, 9841-9852.

[30] Schüring, A.; Auerbach, S. M.; Fritzsche, S.; Haberlandt, R. *J. Chem. Phys.* **2002**, *116*, 10890-10894.