at the University of Freiburg

Master Thesis

February 24, 2021

Author:

Wilkin Wöhler

Supervision:

Tanja Schilling

Abstract

A raw writing of hard sphere nucleation, a simulation to measure quantiteis, and a analysis of data generate by means of the simulation. A test citation: [1]

Contents

List of Figures

List of Tables

1 Introduction

1.1 Hard sphere system

The Hard Sphere system is the simplest model of a fluid including interactions between the single particles. Its well known potential between particles i and j reads:

$$V(r_{ij}) = \infty \cdot \Theta(\sigma - r_{ij}) \tag{1.1.1}$$

In this equation r_{ij} indicates the distance between the two particles, σ is the diameter of the Hard Spheres and Θ is the Heavyside function.
While the ideal gas model without pair interactions already makes it possible to derive famous equations as $pV = NkT$, it does not include phase transitions yet. But these can be observed when granting the particles to take up space. Because it is the simplest model and it is well feasible for computer simulations the Hard Sphere system is very well suited to study basic properties of phase transitions.

Compared to experiments where similar systems can also be realized, general properties of the system at hand can be varied very precisely without much effort and position data of the single particles can be extracted easily as well, because they naturally are required for the simulation.

On the downside computer simulations are much more constraint in their size, but with todays computational possibilites system of the order of 1 million particles become tractable, and such computer simulations become a powerfull tool to study also phase transitions in simple systems.

The beginning of such simulations actually dates back to the beginning of electronic computer technology (cite Alder and Wainwright 1959). Since then more algorithms to increase efficency have been elaborated, and technology advanced giving today the possibility of studying large systems.

1.2 metastable fluid/ phase diagram

The equation of state for the simple Hard Sphere system has various approximations, (cite an overview paper), The probably most common approximation due to its simplicity is the Caranhan-Sterling

approximation:

$$Z = \frac{1 + \eta + \eta^2 - \eta^3}{(1 - \eta)^3} \tag{1.2.1}$$

(cite https://aip.scitation.org/doi/10.1063/1.1672048) It approximates the compressibility factor Z depending on the packing fraction η for the Hard Sphere fluid.

For the stable brach after nucleation a common approximation is given by the Alamrza equation of state(cite https://aip.scitation.org/doi/full/10.1063/1.3133328).

$$\frac{p(v - v_0)}{k_B T} = 3 - 1.807846y + 11.56350y^2 + 141.6y^3 - 2609.26y^4 + 19328.09y^5 \tag{1.2.2}$$

where p is the pressure, v is the volume per particle $v_0 = \sigma^3/\sqrt{2}$ is the volume per particle at close packing, including the diameter of the spheres σ, and $y = p\sigma^3/(k_B T)$, where k_B is the Boltzman constant and T is the temperature of the crystal.

From these two equations of state we can draw the phase diagram: (include grapic of phase diagram.)

The chemical potential difference between the two equations of state can be calculate from the difference between the two equations of state.

Eventhough not further discussed in this thesis it might be said that for polydisperse radii the phase diagram becomes even richer as show for example in https://journals.aps.org/prl/pdf/10.1103/PhysRevLett.91.06830

1.3 Classical nucleation theory :/

Classical nucleation theory (CNT) has been proposed look who did this the first time and since then multiple times modified to describe various types of systems. Eventhough its predictions often deviate far from experimental results. look if tanja cited someone else here. Either way it can provide a reference to know what to expect roughly and also it can be checked by the simulation data.

CNT assumes that a spherical crystalite may form in the liquid with properties of the bulk crystal while the fluid remains with the properties of the bulk liquid. The difference in the free energy landscape is given by a surface and a volume term. The first arises from the surface tension γ between the fluid bulk and solid bulk phases. The second comes by the difference in chemical potential $\Delta\mu$. The whole expression reads:

$$\beta\Delta G(R) = 4\pi R\gamma - \frac{4}{3}\pi R^3 \rho \Delta\mu \tag{1.3.1}$$

Where ρ is the particle density of the solid phase.

The free energy barrier can be sketched like this (maybe sketch it for som mu and gamma).

As can be seen it has a maximum at R_{crit} the critical radius. If the radius of a cluster surpasses the critical radius, it is likely to continue to grow until it incorporates all available fluid. It is given by:

$$R_{crit} = \frac{2\gamma}{\rho\Delta\mu} \tag{1.3.2}$$

Furthermore the height of this barrier can be calculated to be

$$\beta\Delta G(R_{crit}) = \frac{16\pi\gamma^3}{3\rho^2(\Delta\mu)^2} \tag{1.3.3}$$

In the classical picture now we look at an activated process which has an rate given by the Arhenius law:

$$k = c\exp\left(-\frac{k_B T}{\Delta E}\right) \tag{1.3.4}$$

$$\Leftrightarrow \quad k = c\exp -\beta\Delta G(R_{crit}) \tag{1.3.5}$$

As the constant c is not further defined the absolute nucleation rate in this picture is not predicted but only set into realtion to ther rates. Check carefully in how far this is correct!
Given the equation of state for the liquid and fluid phase from section 1.2, and taking a literature value for γ cite the refernce to which the value corresponds we can calculate R_{crit} for various densities. Plot of r_{crit} for different densities

1.4 Memory including approaches

CNT assumes a Markovian system, which means that it diffuses through a free energy landscape without granting it memory. But it has been shown by Kuhnbold ... cite Anjas Lennard-Jones System that for a Lennard-Jones system memory effects can not be neglected if an acurate desciption is desired. Such it must be assumed that also for the Hard Sphere system memory effects may be present.

To analyze such memory effects a framework by Hugues Meyer(cite Hugues) has been elaborated. In it a memory kernel for coarse-grained observables by means of projection operators is defined together with an efficent algorithm to calculate such.
With the derived memory kernel an equation of motion for the observable can be dervied, ressmbling mostly the Generalized Langevin equationbu is called non-stationary Generalized Langevin Equation because the memory kernel can depend not only on $K(t-\tau)$ but instead on two times $K(t,\tau)$:

$$\frac{dA_t}{dt} = \omega(t)A_t + \int_0^t d\tau K(\tau, t)A_\tau + \eta_t \quad , \tag{1.4.1}$$

For the Markovian process the memory kernel is approximately given by a Dirac delta functional $\delta(\tau - t)$, in which case the Langevin equation is recovered.

1.5 Computer Precision

The precision of computer certainly influences the outcome of simulations. (talk with fabian if he found any papers reagarding varying results fromvarying precision over time)
And it should always be kept in mind that the simulation only approximates the real world. Even smallest variations in the last digits of positions, changes the simulation radical after a certain number of steps. This comes by the fact that we face a complex system with chaotic behaviour, which means that even small variations will grow exponentally until the system has nothing in common anymore. Look if you can visulaize such an behaviour by As it can be seen after x timesteps the two system have radicallly changed.

1.6 Comparsion to Real world experiments

Since the 1950s ? people have systhesized hard sphere like systems in the lab. Today a whole zoo of systems is known. All of these system have in common, that the hard spheres are in a bath of a fluid, which surrounds them. This fluids density and optical refractive index shhould match the density and optical refractive index of the hard spheres to prevent segmentation and to enable optical measurements of the system. maybe elaborate a little more on why each of the two is necessary

The absence of the bath in simple hard sphere simulations is probably the largest difference to the hard sphere systems in the laboratory. It has been argued that the difference can be circumvented by normalizing with a diffusion length or time, but a discussion on the possibility of hydrondynamic effects changing the behaviour of the lab system compared to simulations is ongoing at the moment.

For simulations it is much harder to include the bath becuase it introudces a multiple number of particles, making it very slow to simulate large systems.

2 Simulation details

During the coarse of the master thesis an event driven molecular dynamic (EDMD) simulation code has been elaborated. The choice to use the EDMD approach is taken because interst in the actual dynamics of the system were desired. This means that simulations probing the phase space of the system instead of the dynamics, like Monte Carlo (MC) simulation schemes, are not suited.
Furthermore the discontinous potential of the hard spheres is an obstacle not easy to face in regular molecular dynamics (MD) schemes, where the Newtonian equation of motion for the particles is numerically integrated.
The EDMD approach on the other site actually requires these discontinuities as will be discussed in the following sections, together with some details of the program.

2.1 Algorithm and Simulation details

In this section we will highlight the main differences to regular MD simulations, as they are the main tool to otherwise probe the dynamics of the system. Furthermore we will stick to the hard sphere example when discussing the EDMD simulations, but it can be kept in mind that the EDMD approach also allows to simulate particles with other potentials as long as the potentials are only containing step functions.

The decisive difference between EDMD simulations and regular MD schemesis that, instead of evaluating all pair and external forces on each particle and then evolving the whole system to the next time step, EDMD simulations do not have a predefined time step, but the system is evolved from one event to the next one. An event in this context is defined as the time where the next collision in the whole system takes place.

The event prediction algorithm is follows closely the approach proposed by Bannerman et. al [2] which will be discussed in the next section.

2.1.1 Event driven molecular dynamics (EDMD)

For the prediction of events in EDMD simulations an overlap function $f_{ij}(t)$ between particles i and j is defined, where the squared quantities are used merely because they are easily accessible.

$$f_{ij}(t) \coloneqq |\vec{r}_j(t) - \vec{r}_i(t)|^2 - \sigma^2 \tag{2.1.1}$$

$$\left|\begin{array}{ll} \text{with} & \vec{r}_i(t) = \vec{r}_i(t_0) + (t - t_0)\ \vec{v}_i(t_0)\,, \\ & \Delta t \coloneqq t - t_0\,, \\ & \vec{v}_{ij}(t) \coloneqq \vec{v}_j(t) - \vec{v}_i(t)\,, \\ & \vec{r}_{ij}(t) \coloneqq \vec{r}_j(t) - \vec{r}_i(t)\,, \\ \Leftrightarrow & \vec{r}_{ij}(t) = \vec{r}_{ij}(t_0) + \Delta t\ \vec{v}_{ij}(t_0) \end{array}\right. \tag{2.1.2}$$

$$f(t) = (\vec{r}_{ij}(t_0) + \Delta t\ \vec{v}_{ij}(t_0))^2 - \sigma^2 \tag{2.1.3}$$

$$f(t) = |\vec{r}_{ij}(t_0)|^2 + \Delta t^2\ |\vec{v}_{ij}(t_0)|^2 - 2\Delta t\ \vec{r}_{ij}(t_0) \cdot \vec{v}_{ij}(t_0) - \sigma^2 \tag{2.1.4}$$

The overlap function has the property that it is negative for two particles being closer than their diameter, 0 for at collision and positive if neither overlapping nor touching. The calculation of the next collison thus is to calculate the roots of eq. 2.1.4.

Solving for Δt with $|\vec{r}_{ij}(t_0)|^2 \coloneqq rr$, $|\vec{v}_{ij}(t_0)|^2 \coloneqq vv$ and $\vec{r}_{ij}(t_0) \cdot \vec{v}_{ij}(t_0) \coloneqq rv$ is rather trivial:

$$0 = rr + vv\ \Delta t^2 - 2rv\ \Delta t - \sigma^2 \tag{2.1.5}$$

$$\Leftrightarrow \quad 0 = \Delta t^2 - \frac{2rv}{vv}\ \Delta t + \frac{rr - \sigma^2}{vv} \tag{2.1.6}$$

$$\Leftrightarrow\ \Delta t = -\frac{rv}{vv} \pm \sqrt{\left(\frac{rv}{vv}\right)^2 - \frac{rr - \sigma^2}{vv}} \tag{2.1.7}$$

But a caveat when executing on a floating point machine is present as can be seen when considering which solution is of interest. As for a possible collision it is necessary that the two particles move towards each other we can conclude that the scalar product is required to be negative $rv < 0$, because otherwise the particles are already moving away from each other.

Also the quadratic formula has two solutions, corresponding to the entry and the exit of the overlap. Because the entry has to be prior to the exit, we further conclude that interest lies on the smaller solution that is:

$$\Delta t = \frac{-rv - \sqrt{(rv)^2 - vv(rr - \sigma^2)}}{vv} \tag{2.1.8}$$

Now for the case where the distance of the spheres is already close to the diameter of the spheres we find $(rv)^2 \gg (rr - \sigma^2)$, which results in a cancelation of two large numbers leaving a small number. Floating point number operations are inherently bad suited because they tend to large inaccuracy in this case. Rewriting eq. 2.1.8 by making use of the third binomial formula look if this is fine to write. leads to:

$$\Delta t = \frac{(rr - \sigma^2)}{-rv + \sqrt{(rv)^2 - vv(rr - \sigma^2)}} \tag{2.1.9}$$

Comparably eq. 2.1.9 does not contain a cancelation of the type seen before and such is better suited for the use in a computer simulation. cite Goldberg '91

The event prediction algorithm proposed by Bannermann[2] works by differentiating 4 cases:

1. If $rv > 0$ the particles move away from each other leading to a collision time of $\Delta t = \infty$.

2. If $rr < \sigma^2$ an overlap is present resulting in an immediate collision time of $\Delta t = 0$

3. If $(rv)^2 - vv(rr - \sigma^2) \leq 0$ the two particles miss each other, including touching without momentum transfer, resulting in a collision time of $\Delta t = \infty$

4. If none of the before is given the particles collide and Δt is calculated by eq. 2.1.9.

All collision times for a particle are then stored in a queue sorted by event time called particle event list (PEL). From the PEL the first entry is then passed to the global FEL.
This procedure initally takes place for all particles to set up the system and later on for particles involved in an event after its execution.

As will be discussed in section 2.1.2 some redundant calculations have been omitted in the actual simulation program, as well as a cell system added to reach $\mathcal{O}(N)$ computation time, in turn introducing a new event type for cell changes.

A further detail to take care of is the possibility of scheduled events which have bcome invalid due to a earlier collision of the one of the particles. This is handeled by assigning an interaction count to each particle and then store this at precalculation time with the event. When the event comes up, and the interaction count of one of the particles has increased in the meantime, the event is said to be invalidated. Depending on which particle had an event in the meantime the invalidation either causes no action or a recalculation of new events.

2.1.2 Details of the Implementation

As the simulation code is based on an earlier Monte Carlo Code for hard spheres a complete walk through the whole progam would become quite extensive. Such we will focus on key points to understand the details of the simulation.

Event struct

We start with the basic *Event* struct which includes 6 entries as shown in tab. 2.1.1. The type of *time*

Datatype	Name of entry
(timeType)	time
(int)	event_type
(particle*)	particle
(void*)	partner
(int)	particle_count
(int)	partner_count

Table 2.1.1: Content of the *Event* struct.

(timeType) is usually set to double. The *time* variable itself represents the time for when the event is scheduled.
The *event_type* variable is either set to 0 or 1 and indicates if the event is a cell transfer or a collision of two particles.
The *particle* variable is a pointer to the particle for which the event has been pre calculated, while the *partner* variable is set to be a void pointer. Such it is possible to either interpret it as a particle pointer for the collision type event or as an interger pointer to the index in the current cells' neighbours list for transfer events.
In the last two rows the interaction counts for particle and partner are listed as well. As the destination cell in a transfer event does not require an interaction count, the *partner_count* variable is only used for collision events.

The *event* struct is used for all events throughout the simulation. For read and write operations with the HDF5 file format, the struct *event_data* is available which uses only indexes instead of pointers.

Particle class

The *Particle* class is comparably to the one from the MC code basis. Its MC related variables have been removed and additional key variables and concepts will be discussed in the following:

First a vector storing events called *backupEvent* has been added to make it possible to store events from the pre calculation for the case of the first event being invalidated. The idea of reusing events is discussed in many publications, for example that the memory cost increases onyl moderately with more backup events while the speedup does not increase much for more than two stored events [3]. It also has been argued that the added complexity can not account for the increase in efficency[4]. Eventhough in the own simulations a decrease in calculation time of more than 10% was observed and the cost of complexity was seen as moderate. The difference might be explained by the fact that the systems under consideration in this thesis have a rather large particle density, leading to more invalid collsions.

In the context of reusing pre calculated results, it should also be mentioned that after a cell transfer the recalculation of events can be restricted to partner particles only in the new neighbouring cells, leading to only 1/3 of the calculation time in this case. But as mentioned systems under consideration are mostly rather dense and such the number of transfers is often at below 5%. Thus the increase in efficency was assumed to be to costly on the complexity side, and not implemented. Eventhough for sparse systems, it might make sense to include an *updatePEl* routine.

Also key differences to the former MC Particle type are the variables *total_interactions* and *particle_delayed_time*. The first is the variable for book keeping of interactions, while the second represents the event driven character of the simulation. Because each particle only moves on purely ballistic trajectories until an event occurs, it is not necessary to keep all particle positions and velocities synchronized in time. Quite on the contrary it would mean executing extra operations together with summing rounding errors by each floating point operation.

Because sometimes it is desired to have the whole configuration at one point of time, the *transferToTime()* function of the particle provides the possibility to take the particle into the present. This is necessary soon as measurements are performed on the system, including snapshots.

As mentioned before the system behaves chaotic even under slightest changes like a rounding error from an extra floating point operation. A result of this is that measuring at different rates during a simulation changes the simulation trajectory quite a bit. It has been observed that such a system may keep close to the undisturbed trajectory for about 50-100 events/particle. As it is of desire to measure quantities and take snapshots without disturbing the simulation, the simulation program makes employs copies of the configuration being costly in terms of memory but making simulation resets or higher sampling rates at interesting points possible within a defined trajectory.

The structure is as follows: The first copy is rewritten with an image of the working configuration just

before any measurement. The working trajectory iteself is then disturbed by the measurement, and afterwards replaced with its state before the measurement from the backup configuration.
The second copy actually includes the full simulation state, while the first only includes the particle cofiguration. This second one might be used to save a state during the simulation and reset to just the same point at any later time.

The *Box* class

The box of the simulation stayed mostly the same as in the previous MC code. One chanege is the array *neighbours_lookup* which has been added. It contains the indices for the cells' *neighbours* array pointing to cells that share their surface. It is used to identify which cell a particle has to be transfered to during a cell transfer event.

Furthermore the *Update* routine now takes care of all quantities depening on the length of the box, making the *rescale* routine a simple rescaling of the edge lengths with an additional *Update* command.

The *Scheduler* class

While the afore mentioned parts of the program are necessary for the EDMD program, the *Scheduler* class contains the most distinct parts of the program. It keeps track of all events to come, predicts new events and orchestrates the execution of the events. The essential functions are discussed in the following subsections while some basic properties are shortly highlighted here.

First of all the *Scheduler* holds the Future event list (FEL) in which at least one event per particle is stored. As discussed within section 2.1.2 the simulation is capable of saving the complete state of a trajectory, including all pre calculated events. For this purpose an array of *Events* is available.
Furthermore the *Scheduler* includes the *gloabl_time*.
Important for the efficency is the pre allocation of all arrays used within the prediction calculations, as the number of executions for the collision prediction routine is about $\frac{30}{\text{particle}\cdot\text{step}}$ accounting to a few billion function calls during a small simulation.

Scheduler::predictTransfer()

As the name suggests this function predicts the next cell transfer of a particle due to its movement. For this it calculates the position of the particle at global time, which for a valid state of the simulation always lies within its cell. By transforming the momentary position of the particle from the global

coordinate system to the coordinate system of the cell and taking into account the periodic boundary conditions, we can write for each dimension i the equations

$$t_{i1} = -\frac{r_i}{v_i} \quad \text{and} \quad t_{i2} = \frac{l_i - r_i}{v_i} \tag{2.1.10}$$

which describe the times when the particle pierces the cell's left and right boundaries. A negative time corresponds in this case to a boundary crossing in the past, a time comparable to 0 means that the particle is on the edge of its cell and a positive time means that the boundary crossing lies in the future. By going through the different possible cases for t_1 and t_2 we find the resulting next crossing time for each case as shown in tab. 2.1.2.

t_1	t_2	Result	Case
$>$	$>$	invalid	-
$>$	$=$	$t_{\text{crossing}} = t_1$	0
$>$	$<$	$t_{\text{crossing}} = t_1$	1
$=$	$>$	$t_{\text{crossing}} = t_2$	2
$=$	$=$	invalid	-
$=$	$<$	$t_{\text{crossing}} = t_1$	3
$<$	$>$	$t_{\text{crossing}} = t_2$	4
$<$	$=$	$t_{\text{crossing}} = t_2$	5
$<$	$<$	invalid	-

Table 2.1.2: Possible results for left and right crossing time with resulting choice of next crossing time. $>$, $=$ and $<$ are to be read as for example $t_1 > 0$. The case indicates the case number within the actual simulation.

By collecting the next crossing times for each dimension and taking the minimum of these times the exit time of the particle from its cell is determined.

The return value of the routine is an *Event* where the partner is given as an address to the box' *neighbours_lookup*. The index lies between 0 and 5, corresponding to the 6 possible neigbhour cells sharing a surface with the current cell of the particle. Each valid case represents a distinct neigbour cell and the index within the cells *neighbours* array is clearly defined by the cell setup routines and is shown in tab. 2.1.3.

dimension	boundary	case	index
x	front	0	12
	back	1	13
y	front	2	10
	back	3	15
z	front	4	4
	back	5	21

Table 2.1.3: Overview of the cells' *neighbours* indices directly sharing a surface for 3 dimensions. As the indices hardly follow any simple pattern they are explicitly noted at this point. Obviously the cell consists of a front and a back boundary in each dimension. The corresponding case matches the one from tab. 2.1.2.

Scheduler::predictCollision()

The prediction of collision times has already been discussed in section 2.1.1. The implementation in the program first calculates all necessary scalar products while accounting for the periodic boundary conditions, and in a second step returns the collision time depending on the case at hand.

The presented algorithm is only valid for single sized particles. If polydisperse systems are supposed to be considered the algorithm has to be adjusted. mayhap do it in the appendix?

As this routine is executed through out the simulation very often it has been tried to optimize its efficency multiple times. For example calculating only necessary results for the next case differentation has been implemented but no significant increase in efficency was recognized and for better readability the prior version has been reestablished. In either case if an optimized way of calculating the results is found it might be useful to use them.<– Not really nice

Scheduler::setupFEL()

This routine fill the FEL of the simulation. For this purpose it iterates through all particles and calls *setupPEL* for each of them. The PEL in turn is set up by predicting the next cell transfer as well as the next collisions with all particles within the 3^d cells directly surrounding the particle. From all predicted events only such with finite times are then written to the *backupEvents* vector corresponding to the PEL of the particle.
For the FEL only the top event of each particle is then used. Because other events from the PEL are able to move on to the FEL ounce written to the FEL an event has to be erased from the PEL.

Scheduler::executeTransfer()

The execution of a transfer event is accomplishd by the event particles *MoveBetweenCells()* routine. The departure cell is taken as the event particles own cell. While the event partner holds the information which of the cell neighbours is the destination cell.

Scheduler::executeCollision()

The outcome of a collision between particle 1 and 2 with corresponding position and velocity can be derived by momentum and energy conervation cite some textbook, or better jsut write down the calculation too be

$$\vec{v}_1' = \vec{v}_1 + \left(\frac{\vec{r}\cdot\vec{v}}{\vec{r}\cdot\vec{r}}\right)\vec{r}_{12} \tag{2.1.11}$$

This equation is not directly dependent on the radius, but only

Scheduler::executeEvent()

2.2 Probe of simulation code

To probe we have to measure known quantities

2.2.1 Diffusive behaviour

Show the diffusive behavior of at least the fluid

2.2.2 Radial distribution function

Show a RDF of the fluid , if possibible with the theoretical cervus-pevick approximation

2.3 Estimate of required resources

2.3.1 Calculation time estimates

GIve some profiling numbers of the simulation Also conclude that missing q6q6 O(N^2), broke the walltime.

2.3.2 File sizes estimates

Show the estimate on the file size

2.4 Produced Data

Overview of produced data with visualized snapshot?

2.4.1 Equilibration steps

Show dependence of equilibration steps on simulation

2.4.2 Initial density

Show dependence of initial density on simulation

2.5 Possible extensions

Highlit possible extenisions which migth be doable to implement.

2.5.1 Varying radius

Vary the radius of the spheres. Rewquirements and thought on that.

2.5.2 Varying mass

Same as with radius, highlight the points in the code requiring a change.

2.5.3 Muliprocessing

Idea on Parrallelizing the code. Probably more advanved, but for larger systems (Really large files) doable. (Actually RAM might become a serious problem).
Besenrein, Samstag 6.3. mittag, 13 h

3 Data Analysis

This is the analysis part

3.1 Diffusion of the lqiuid

This contain analysis of diffusion in the liquid to prove the simulations accuray

3.2 Diffusion of the metastable liquid

This contains Diffusion constants to normalize the rates

3.3 Cluster growth

Cluster growth depending on density

3.4 Tensor of Gyration properties

Well only swamp here, but it can be shown to conclude the swamp.

3.5 ACF largest cluster?

Just in case anything can be seen here

3.6 Nuclection time dilemma

Evaluation of induction time. Prolem with accuracy and precision. Compare methods.

3.7 Induction time by exponential distribution

Obtain exp assumption and best estimator

3.8 Nucleation rate comparison

All Nucleation rates that can be found.-> mayhap ask Hajo.

3.9 Memory Kernels

Memory kernels of systems at various densities. Depends strongly on what is found here

4 Conclusion - Summary

4.1 Conclusion

5 Appendix

.1 A

Bibliography

[1]D. M. Heyes, M. J. Cass, J. G. Powles, and W. A. Evans, "Self-diffusion coefficient of the hard-sphere fluid: System size dependence and empirical correlations", Journal of Physical Chemistry B **111**, 1455–1464 (2007).

[2]M. N. Bannerman, S. Strobl, A. Formella, and T. Pöschel, "Stable algorithm for event detection in event-driven particle dynamics", Computational Particle Mechanics **1**, 191–198 (2014).

[3]M. N. Bannerman, R. Sargant, and L. Lue, "DynamO: A free O(N) general event-driven molecular dynamics simulator", Journal of Computational Chemistry **32**, 3329–3338 (2011).

[4]A. DONEV, S. TORQUATO, and F. STILLINGER, "Neighbor list collision-driven molecular dynamics simulation for nonspherical hard particles.II. Applications to ellipses and ellipsoids", Journal of Computational Physics **202**, 765–793 (2005).