Investigating the role of solvent on the double stranded DNA melting

R.H.A. Fransen

9th May 2011

Abstract

With the knowledge of the structure of the DNA, nowadays new challenges focus on other parameters, describing the behavior, of the DNA. In this study, preliminary simulations are performed on DNA to investigate the role of solvent on melting of DNA. To do so, the software packages VMD and NAMD [1,2] are studied together with the learning of the programming language behind those systems, which is TCL [3]. The simulations verified that the used software is able to describe such systems ($\approx$10.000 atoms) with enough detail to keep the simulations stable. Furthermore, the simulation (which is a 10 base-pair piece of the lambda - phage half-operator DNA [4]) showed the initiation of melting which is indicated by the separation of the two DNA chains. The model was validated by simulating and analyzing water (based on the TIP3P water model) for several systems and temperatures.

Preface

Within the master programme on Mechanical Engineering at the University of Twente, Enschede, the Netherlands, an internship of 15 European Credits must be performed. This is equivalent to 420 hours or three months of work. My internship is carried out at the Universidade Federal de Minas Gerais (UFMG), Belo Horizonte, Brazil under supervision of Prof. João Antônio Plascak. Furthermore, Bruno Barbosa Rodrigues supported me during the internship as being my direct advisor. This study is a preliminary research for the upcoming three years in which he will develop his PhD project. The supervisor from the University of Twente is Prof. Stefan Luding (Multi Scale Mechanics, Faculty of Engineering Technology). Concluding, the Universidade Federal de Sergipe, Aracaju, Brazil also offered their facilities for which we would like to express our gratitude.

Contents

Introduction

The structure of the double stranded Deoxyribonucleic Acid (DNA), see appendix B for a list of acronyms, was discovered by James Watson and Francis Crick (see, for example, [5] and references therein) in 1953. Based on three observations, they managed to construct the structure which is nowadays generally accepted. The first one was the observation that the DNA polymerized through the formation of phosphodiester linkages which indicated that the DNA had a sugar-phosphate backbone. The second indicated that 1) the number of Thymine (T) and Adenine (A) groups and 2) the number of Cytosine (C) and Guanine (G) groups were the same. The final observation was done with the use of X-ray crystallography [6] by Rosalind Franklin and Maurice Wilkins [5] and ended up in three important distances: 0.34nm, 2.0nm and 3.4nm. Watson and Crick figured out that DNA was build with purine-pyrimidine pairs (T-A and C-G) which indicated that the 2.0nm distance was the diameter of the DNA. Furthermore they figured out that the distance between two of those base-pairs was 0.34nm and that it takes 10 base-pairs (3.4nm) to get a complete turn in the helix.

Even though the structure of DNA is well understood, the melting of it is still an unknown process in physics. It is known, however, that many variables, such as salt concentration [7], loop sequence [8] or solvent type [9], can drastically affect its properties, especially the temperature under which this phenomena occurs. Despite the importance of this process for transcription, translation and replication, there is still a lack on the theoretical background that supports the sharp DNA melting profiles observed experimentally. On the other hand, computer simulations (like Weber, G. *et al.* [10]) can give insights on the molecular behavior of the double-stranded DNA and probe the real function of the solvent on the mechanism that drives the melting. In order to try to clarify this mechanism, in the present work a detailed molecular dynamics simulation was performed for the 10 base pairs λ - phage half-operator DNA (PDB code 1D20 [11]) over a range of temperatures raising from 300K up to 350K for a system containing the piece of DNA, water and salt which is neutrally charged. The simulation approach for this system is shown in chapter 1 and the system itself is explained in more detail in chapter 2. The simulations are performed with the use of several software packages, see appendix C. The model is validated by checking the properties of water, see chapter 3, which is the main focus of this study. The study concludes with providing preliminary results on the DNA simulation (chapter 4) and acknowledgments (chapter 6).

1 Simulation protocol

Simulating the DNA, surrounded by water and/or salt ions, requires a three step protocol based on a protocol developed by Shields *et al.* [12]. This protocol includes minimization (§1.1), equilibration (§1.2) and heating (§1.3) and guarantees that the heating starts from a stable situation.

1.1 Minimization

Minimizing the energy of the system is mandatory in retrieving the state static equilibrium. The conjugate gradient method, as described in §2.2.1, is a way to search for this state of lowest energy in a iterative way. To do so, the total energy of the system should be dominated by a second order polynomial [13]. This is the case since the kinetic energy is kept zero (the conjugate gradient method repositions the molecules without applying a velocity) which implies that the total energy is only build from the potential energy as described in §2.1.2. Applying this method on the system at hand will reposition the molecules (note that this method cannot break covalent bonds) based on a state of less energy. However, since the potential is more complex than just a second order polynomial and the system is big and build with different types of molecules, it is possible that the method results in a local minimum instead of the global minimum.

1.2 Equilibration

After minimization the system is released and run for a relatively long time with constant temperature and pressure. In this case, the proposed Verlet algorithm (§2.1.1) is used in combination with the force calculation (§2.1.2) and the additional statistical models (§2.2) are applied to the system to decrease the energy even further. The total energy energy will then converge to a lower level; the optimal locations of all the molecules is then reached.

1.3 Heating

With the minimization and equilibration performed, the system is ready to heat up. Heating is performed by manually increasing the Langevin temperature (see §2.2.2) by an fixed increment and then applying a new equilibrium simulation at this temperature. In the preliminary DNA simulation, the increment was 5K per 50,000 time steps which is equal to 0.1ns per 5K.

2 The DNA - a molecular dynamics approach

The introduction described the basics of the complex 3D structure of a double stranded DNA. The λ - phage half-operator is an example of double stranded DNA and is used for many types of experiments on DNA, like the one described by Gomes, L.S. *et al.* [14]. Just a small piece of this DNA type is used in simulations for the purpose of saving computational time. One can imagine that computing a DNA model with ≈30.000 base pairs [14] is very computational expensive. Therefore, just a 10 base-pairs sample, as provided by the Protein Data Bank [15], of this DNA is considered in this study. This chapter gives an introduction to the molecular dynamics approach by explaining the general approach step by step (§2.1), the additional theoretical models used specific for the problem at hand (§2.2). Furthermore, it describes the steps in pre-processing (§2.3) and post-processing (§2.4) as performed in the used software packages.

2.1 The molecular dynamics approach

The second law of Newton is the physical base for solving a system with the use of molecular dynamics. It states that the sum of all forces $\sum f_{a,i}$, see appendix A for the nomenclature, is equal to the mass m_a times the acceleration $\ddot{r}_{a,i}$. This is the general case and can be applied to all kinds of systems where quantum mechanics is neglectable. Examples are constructions, tools or granular systems. In what follows, the subscript i implies the i^{th} direction in space, every dot on top of a vector is time derivative and the ',' is used to distinguish spatial subscripts from particle indexes. When this law is applied to a granular, or in this case, molecular system, all of the 'particles' must fulfill this balance in every direction in space. Therefore, the second law of Newton is applied to each individual particle in every direction. The summation thus sums the different types of forces acting on particle a in direction i and is given by

$$\sum f_{a,i} = m_a \ddot{r}_{a,i}. \tag{1}$$

The new locations of all particles can be found by integrating this set of $3N$ coupled equations twice in time, where the N is the number of particles, which, in the system at hand, equals the number of atoms. However, since the size of the system is, in general, significant (≈10.000 particles) and the forces are rather complex, the set of coupled equations is typically solved in an iterative way. This section shows the used numerical integration method (§2.1.1) and explains the force calculation (§2.1.2).

2.1.1 Numerical time integration

Numerical time integration is used in many types of problems in Physics and there are several algorithms available. One of these algorithms is the so called 'Verlet' [16] algorithm which makes use of a discretized time. This discretization is done by splitting the continuous time domain with the use of a time step δt. The value of this time step is, in general, for DNA simulations, equal to 2fs [17, 18]. The method is executed for each atom on each time step and it is based on two Taylor expansions (equations 2 and 3).

$$r_{a,i}(t+\delta t) = r_{a,i}(t) + \delta t \dot{r}_{a,i}(t) + \frac{1}{2}\delta t^2 \ddot{r}_{a,i}(t) + \mathcal{O}\left(\delta t^3\right), \tag{2}$$

$$r_{a,i}(t-\delta t) = r_{a,i}(t) - \delta t \dot{r}_{a,i}(t) + \frac{1}{2}\delta t^2 \ddot{r}_{a,i}(t) + \mathcal{O}\left(\delta t^3\right). \tag{3}$$

By adding and reshuffling the above equations, the following equation is retrieved

$$r_{a,i}(t+\delta t) = 2r_{a,i}(t) - r_{a,i}(t-\delta t) + \delta t^2 \ddot{r}_{a,i}(t) + \mathcal{O}\left(\delta t^3\right), \tag{4}$$

while subtracting the equation 2 from equation 3 results in

$$\dot{r}_{a,i}(t) = \frac{r_{a,i}(t+\delta t) - r_{a,i}(t-\delta t)}{2\delta t} + \mathcal{O}\left(\delta t^3\right). \tag{5}$$

Note that the acceleration on the current time step is the only new parameter which is needed to perform the calculation. This acceleration is retrieved by solving the second law of Newton in the form for molecular dynamics, see equation 1. A more detailed explanation of this calculation is given in §2.1.2.

2.1.2 Force calculation

As stated in the previous section, getting the accelerations is the essential step in solving the system. To do so, the second law of Newton is rewritten as

$$\ddot{r}_{a,i}(t) = \frac{\sum f_{a,i}(t)}{m_a}, \tag{6}$$

from which the forces are the 'only' unknown. These forces are calculated by solving equation 7, which states that the sum of all forces for a certain particle a in direction i is equal to the negative gradient of a scalar potential energy function U_a at time step t. Note that the time step indication is omitted in the notation. Thus the equation is

$$\sum f_{a,i} = -\nabla_i U_a, \tag{7}$$

where this potential can be split in a bonded and non-bonded part

$$U_a = \sum U_a^{bonded} + \sum U_a^{non-bonded}. \tag{8}$$

The bonded one consists of three terms

$$\sum U_a^{bonded} = U_a^{bond} + U_a^{angle} + U_a^{dihedral}, \tag{9}$$

and the non-bonded one of two terms

$$\sum U_a^{non-bonded} = U_a^{LJ} + U_a^{elec}. \tag{10}$$

U_a^{bond} is the first term in the bonded potential and describes the harmonic vibrational motion between the (a, b)-pair of covalently bonded atoms; a pair of atoms sharing a pair of electrons and making a rigid connection within a molecule. The bond is described by

$$U_a^{bond} = k(r_{ab} - r_0)^2, \tag{11}$$

where k denotes the spring stiffness of the bond, $r_{ab} = \sqrt{(r_{a,i} - r_{b,i})^2}$ is the current distance between the two particles and r_0 is the equilibrium distance between two atoms. The second term in the bonded potential is

$$U_a^{angle} = k_\theta(\theta - \theta_0)^2 + k_{ub}(r_{ac} - r_{ub})^2, \tag{12}$$

and denotes the 3-body angular bond potential between the atoms a, b and c; it describes the potential energy related to the angular vibrational motion for three covalently bonded atoms. In this equation k_θ denotes the angle constant, θ is the angle in radians between the vectors $r_{ab,i} = r_{a,i} - r_{b,i}$ and $r_{cb,i} = r_{c,i} - r_{b,i}$ and θ_0 is the equilibrium angle. The second term of the equation describes the Urey-Bradley potential [19] with: k_{ub} as the as Urey-Bradley constant, $r_{ac} = \sqrt{(r_{a,i} - r_{c,i})^2}$ as the distance between the a and c particle and r_{ub} the equilibrium distance. The last term in the bonded potential is the 4-body dihedral angle potential [16] and describes the angular spring over a sequence of 4 covalently bonded atoms: a, b, c and d. For example, consider a hydrogen peroxide molecule (`H-O-O-H`), hold both of the oxygen atoms and try to twist both of the hydrogen atoms around the `O-O` bond; the total encountered resistance is the angular spring as mentioned before. Note that this potential is zero for all water molecules; there are just three atoms in such a molecule. The equation is given by

$$U_a^{dihedral} = \begin{cases} k(1 + cos(\xi\psi + \phi)) & \text{if } \xi > 0 \\ k(\psi - \phi)^2 & \text{if } \xi = 0, \end{cases} \tag{13}$$

in which ξ is a constant positive integer indicating periodicity ($\xi = 0$ indicates no periodicity), k is a multiplicative constant (dimension changes if $\xi = 0$), ψ is the angle in radians between the (a, b, c)-plane and the (b, c, d)-plane and ϕ is the phase shift angle. The first non-bonded potential is the Lennard-Jones potential [16]

$$U_a^{LJ} = \varepsilon \left[\left(\frac{r_m}{r_{ab}} \right)^{12} - 2 \left(\frac{r_m}{r_{ab}} \right)^{6} \right], \tag{14}$$

and describes the weak dipole interactions between all (a, b)-atom pairs in the system. Note that the computational time of this, and the other non-bonded potential, is of $\mathcal{O}(N^2)$ while the computational time of the bonded potentials is of $\mathcal{O}(N)$. The equation makes use of ε which is the depth of the potential well, r_m as the distance at which the potential reaches its minimum (with the value of $-\varepsilon$) and r_{ab} as the distance between the current two atoms as described before. The last term describes the electrostatic interaction [16] between the two atoms a and b

$$U_a^{elec} = \varepsilon_{14} \frac{C q_a q_b}{\varepsilon_0 r_{ab}}. \tag{15}$$

This interaction is repulsive if the charges of the atoms are of the same sign and attractive if the signs are different. Here, r_{ab} is the current distance between the two atoms, C is Coulomb's constant, ε_0 is the dielectric constant, q_a and q_b are the charges of the atoms a and b and ε_{14} is a dimensionless scaling factor with a value $0 \leq \varepsilon_{14} \leq 1$. By using this scaling factor, computational time is saved by not counting the 3 closest atoms. In this way the effects of the dihedral potential are combined with the electrostatic potential. The DNA is instable from origin due to this potential; the sugar-phosphate backbone is strongly negatively charged and results in a strong repulsive force between the two chains.

2.2 Statistical mechanical models

The previous section described the basic calculations in a molecular dynamics simulation. For the system at hand, several other theoretical models are used in addition to the basic theory. This section describes the use of the conjugate gradient method, constant temperature control, periodic boundary conditions and constant pressure control.

2.2.1 Conjugate gradient method

The potential is described by quadratic terms (the bond potentials), a Lennard-Jones potential and an electrostatic potential. Therefore, the conjugate gradient method [13] is used in the minimization steps (see §1.1) to approach the state of minimum energy in an iterative way. To do so, the minimum of the energy is found by solving a linear system: $\mathbf{Ar} = \mathbf{b}$ where $\mathbf{r}$ is a n vector with all the degrees of freedom (which is $3N$) in the system, $\mathbf{A}$ the $n \times n$ square, symmetric and positive-definite matrix of the linear dependence of the solution $\mathbf{b}$ (which is a n vector). This system is solved with

$$\mathbf{r}_{j+1} = \mathbf{r}_j + \alpha_j \mathbf{d}_j, \tag{16}$$

in which $\mathbf{r}_{j+1}$ is the next approximation, $\mathbf{r}_j$ the current approximation and $\mathbf{d}_j$ a search vector which is weighted by α_j. Note that all vectors and matrices in this section are emphasized by bold characters. By defining the search vector as

$$\mathbf{d}_{j+1} = \boldsymbol{\mathcal{R}}_{j+1} + \beta_{j+1}\mathbf{d}_j, \tag{17}$$

which is composed by the residual $\mathcal{R}$

$$\boldsymbol{\mathcal{R}}_{j+1} = \boldsymbol{\mathcal{R}}_j - \alpha_j A\mathbf{d}_j, \tag{18}$$

and β_{j+1}

$$\beta_{j+1} = \frac{\boldsymbol{\mathcal{R}}_{j+1}{}^T \boldsymbol{\mathcal{R}}_{j+1}}{\boldsymbol{\mathcal{R}}_j{}^T \boldsymbol{\mathcal{R}}_j}, \tag{19}$$

which is achieved after Gram-Schmidt conjugation [13]. This β_j is 0 for $j = 0$ which results in an initial condition of

$$\mathbf{d}_0 = \boldsymbol{\mathcal{R}}_0 = \mathbf{b} - \mathbf{Ar}_0. \tag{20}$$

The weight function α_j is given as

$$\alpha_j = -\frac{\boldsymbol{\mathcal{R}}_j{}^T \boldsymbol{\mathcal{R}}_j}{\mathbf{d}_j{}^T \mathbf{A}\mathbf{d}_j}, \tag{21}$$

and the method takes n iterations to solve the system which is the number of degrees of freedom in the system ($3N$) [13].

2.2.2 Constant temperature control

Temperature calculations are based on the kinetic energy [16] of the particles and thus the speed at which the particles move. There will be fluctuations in the velocities since the Verlet algorithm (see §2.1.1) is an iterative solver and there are solvate interactions between the different molecules which are not covert by the potentials. A control on the temperature is thus mandatory. This control can also be used to heat the system by manually increasing the kinetic energy in the system. Langevin dynamics [20–22] is used to control the kinetic energy of each atom in the system. This method adds a virtual frictional damping to Newton's second law, which is then given by

$$m_a \ddot{r}_{a,i} = \sum f_{a,i} - m_a \gamma_a \dot{r}_{a,i} + R_{a,i}, \tag{22}$$

in which γ_a is a frictional constant and the last term describes the random forces due to the solvate interaction. Solving the second order partial differential equation of equation 22 is done by splitting it in two first order ordinary differential equations, which are given by

$$\dot{r}_{a,i} = s_{a,i}/m_a, \tag{23}$$

$$\dot{s}_{a,i} = \sum f_{a,i}(t) - \gamma_a s_{a,i} + R_{a,i}(t), \tag{24}$$

and are the so-called equations of motion.

2.2.3 Periodic boundary conditions

Periodic boundary conditions are often applied in molecular dynamics simulations. These conditions 'mirror' the system at hand in all directions around the system. By applying these mirrors, particles in the 'left' side of the system also interact with particles on the 'right', this applies to all directions. These conditions are initiallized in the beginning of the simulations are are fixed if an NVE (constant number of particles, volume and energy) ensemble is modeled but they may change if an NpT ensemble is modeled. The applied pressure will then change the volume of the system.

2.2.4 Constant pressure control

Controlling the pressure is done with use of the Langevin piston Nose-Hoover [23] method which adds another term to the second law of Newton;

$$m_a \ddot{r}_{a,i} = \sum f_{a,i} - m_a \gamma_a \dot{r}_{a,i} + R_{a,i} - \mathcal{S}(\ddot{e}, \dot{e}, r_{a,i}), \tag{25}$$

in which

$$\mathcal{S}(\ddot{e}, \dot{e}, r_{a,i}) = m_a r_{a,i} \left[\ddot{e} + \dot{e}^2 + \gamma_a \dot{e}\right], \tag{26}$$

which results in the equations of motion

$$\dot{r}_{a,i} = s_{a,i}/m_a + \dot{e} r_{a,i}, \tag{27}$$

$$\dot{s}_{a,i} = \sum f_{a,i} - \dot{e} s_{a,i} - \gamma_a s_{a,i} + R_{a,i}. \tag{28}$$

The added term prescribes a ratio on the change of volume and the volume itself, as given by

$$\dot{e} = \frac{1}{3}\frac{\dot{V}}{V}, \tag{29}$$

and is controlled by prescribing the rate of change of the volume

$$\ddot{V} = \frac{1}{W}\left[p - p_0\right] - \gamma_e \dot{V} + R_e. \tag{30}$$

which is build from the instantaneous pressure p, the target pressure p_0, a 'collision frequency' γ_e and a random 'force' R_e which is taken from a Gaussian distribution with zero mean [20–22]. W is the 'mass' of the piston and has the dimensions of kg/m^4

2.3 Pre-processing

A flowchart, see figure 2.1 [17], is used to implement the DNA in VMD and NAMD. This chart can be used for general problems and has to be specified more for the system at hand; the solvation step is not shown. The implementation starts with reading the Protein Data Bank (PDB) file [4] (§2.3.1), which should be split by segment (§2.3.2). By combining this information about coordinates with structural information, like charges and masses, from a topology file (§2.3.3) a Protein Structure File (PSF) can be generated (§2.3.4). With the PSF and PDB files of the DNA at hand, it is possible to run several functions in VMD. These functions include, for example, solvation (§2.3.5) and ionization (§2.3.6) of the system and result in new PSF and PDB files where atoms of water molecules and/or ions (like natrium and chloride) are included. Using this information, combined with the other information like the temperature range and the parameter file, a configurations file can be written (§2.3.7) which is read and executed by NAMD.

2.3.1 PDB file

The first step in simulating the DNA is building it atom by atom. Therefore, the coordinates of the DNA have to be entered in the software. It is possible to read a PDB file which contains the coordinate information. These files are in general supplied by the Protein Data Bank [15] and in this case the 1D20 DNA [4] was used. A PDB file typically contains a lot of information from which the ATOM information is the most crucial. This part of the file is shown below and table 1 shows the explanation for each column in the file.

```
1       2    3   4  5  6    7       8      9          10    11   12
ATOM    329  C6  DC B  11   2.850   0.310 -13.870  1.00  0.00 C
```

From which can be concluded that this line identifies the 329th atom in the file which is the C6 atom in the eleventh group of the DNA. This group is a Cytosine group located on the first position of the B chain. This is also visualized in figure 2.2(a); the white chain represents the 'A' chain and the yellow chain the 'B' chain from which the first group (the group to which the atom belongs to) is colored green. The atom itself is shown in red.

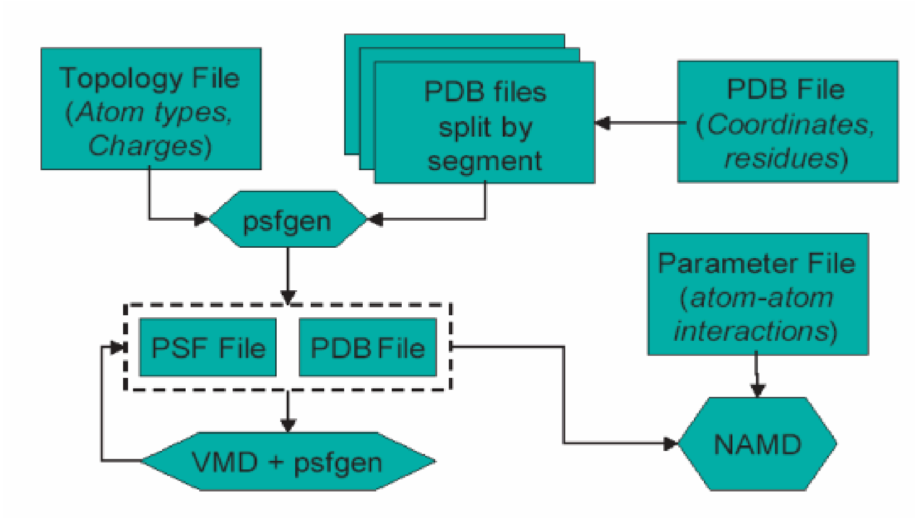


Figure 2.1: Flowchart of input requirements

ID	Input	Function
1	ATOM	Identifies that this line contains ATOM information
2	329	Atom ID, this is the 329th atom in this PDB file
3	C6	Atom type (Carbon), the 6 indicates which carbon it is (topographically spoken)
4	DC	Group type (Cytosine)
5	B	Chain ID (Second chain)
6	11	Group ID (Eleventh group of the DNA)
7	2.850	X-coordinate
8	0.310	Y-coordinate
9	-13.870	Z-coordinate
10	1.00	Occupancy (Used to identify fixed/released atoms)
11	0.00	Beta (Used for Thermo coupling)
12	C	Atom type

Table 1: Brake down of the PDB input file

2.3.2 PDB Splitting

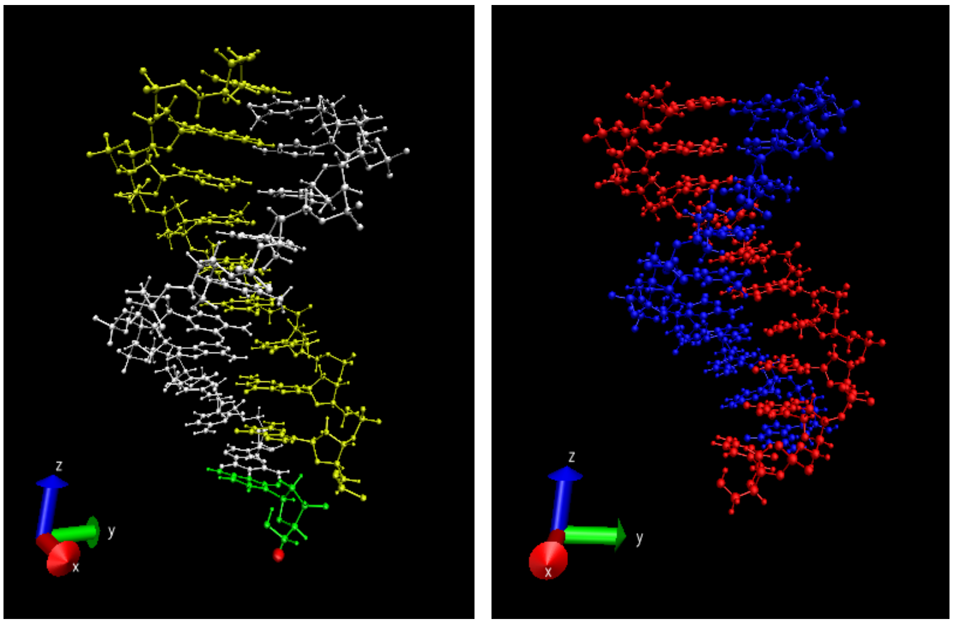
The second step in the flowchart (see figure 2.1) is splitting the PDB files by segments. In the case of a DNA structure, the structure has to be split by the chains. The PDB file contains the needed information for segmentation; the chains are identified by either an ‘A’ or ‘B’. New PDB-files are created for every segment. A representation of split DNA is shown in figure 2.2(b).

2.3.3 Topology file

Before the third step in the flowchart (see figure 2.1) can be taken, a topology file is needed. This file contains any structural and topographical information about nucleic acids and proteins in general. It is supplied by CHARMM [24] and the information about nucleic acids is based on the work of Foloppe and MacKerell [25]. Furthermore, this file comes with another file which is similar: the parameter file. More information on this file in §2.3.7. The topology file includes information about atomic masses and the connections in each residue (Adenine, Cytosine, Guanine or Thymine).

2.3.4 PSF file generation

The fourth step in the flowchart combines the previous two steps: translate the PDB coordinates with help of the topology file into a Protein Structure File (PSF). A ‘dictionary’ is also needed to match the jargon in the PDB files with the jargon in the topology file. The PSF generator supplies this dictionary and matches all the information in a new PSF file.



(a) Brakedown of the PDB input (b) DNA identified by segment

Figure 2.2: Two DNA representations

2.3.5 Add water

This step is not shown in the flowchart but, since DNA cannot exist in a vacuum (due to the strong nevative charged backbones) a solvate must be added. Therefore, water is needed in all directions surrounding the DNA. The `solvate` package is able to do this. This package makes use of an equilibrated TIP3P water cube in a liquid state which determines the place of the added water molecules surrounding the DNA. Nevertheless, it should be noted that the configuration of these molecules is probably far from the lowest energy configuration. Therefore, the system needs to be minimized (§1.1) and equilibrated (§1.2).

2.3.6 Add ions

DNA is strongly negatively charged which can be compensated by adding positive ions. Natrium ions (Na^+) are chosen as the positive charged ions and are used to neutralize the system. Adding more Na^+ ions in combination with Cl^- ions makes it possible to implement salt in the system. The `autoionize` package is used to add the ions to the system.

2.3.7 Write configurations file

With all input parameters being known, the configurations file can be made. This file is the input file which is send to NAMD and contains information from the previous sections. Furthermore, it contains information about the integration, forces, temperature, pressure and other boundary conditions. It is also possible to continue from a previous simulation or to implement a heating protocol in this configuration file.

2.4 Post-processing

NAMD supplies several types of files which need post-processing. For example, the file containing the coordinates of all the atoms on all the time steps is used to get insight in the number of hydrogen bonds in the system that can be used to, to calculate the radial distribution function (RDF) or to measure the density of the system. Matlab is, thereafter, used to generate plots with use of the file containing information on the energy in the system or, for example, to plot the (calculated with VMD) number of hydrogen bonds. This section describes the method of calculating the density (§2.4.1), the RDF (§2.4.2) and the number of hydrogen bonds (§2.4.3). A brief description on the use of Matlab is given in §2.4.4.

2.4.1 Calculation of the density

Calculating the density is essential in investigating phase transitions in the solvate; the density is a function of temperature which should be covered by the used water model. The density calculations are not performed for systems involving DNA since this property is not used as a measurement method to determine other parameters, moreover, a relation between the density of DNA and water is not found and investigating phase transitions within the DNA is not in focus of the study. Calculating the number of hydrogen bonds (see §2.4.3) provides more insight in the behavior of the DNA on high temperatures. The TIP3P water model is used as starting point which is, in general, considered as one of the models which covers such kind of physical behavior less then others. Nevertheless, using the TIP3P model saves a lot of computational time due to its simplicity. Knowing, from basic Physics, that the density of water should reach a maximum near 4°C [26], the model should increase in density before and decrease after this value. Calculating the density is based on counting the number of oxygen atoms within a fixed part of the system. The density of the water systems at the current time step is calculated by applying

$$\rho_w = \frac{N_O * m_{H_2O}}{L * V_b}, \tag{31}$$

in which N_O is the number of oxygens within box V_b at a the current time step, m_{H_2O} the molecular mass in kg/mol and L Avogadro's constant in mol^{-1}.

2.4.2 Calculation of the Radial Distribution Function

The Radial Distribution Function (RDF) provides insight in the relative density of a system and follows the general definition of

$$\rho(r) = \rho g(r), \tag{32}$$

in which $\rho(r)$ is the average density at a distance r; ρ is the average density of the continuum and $g(r)$ the radial distribution function. Furthermore the radial distribution function must fulfill

$$\int_0^\infty \rho g(r) 4\pi r^2 dr = N - 1 \approx N, \tag{33}$$

in which the average density ρ is taken in the units of $1/\text{Å}^3$. This unit is achieved after taking Avogadro's constant (mol^{-1}) and the averaged molecular mass (kg/mol) into account. Multiplying this density by the radial distribution function and integrating it over the volume of a sphere with infinity radius would end up in counting all other particles in the system. From physics one can bound the radial distribution function in several ways. For example, the RDF should return (for $r \to \infty$) unity if it is calculated for a continuum; the averaged density should be equal to the density of the continuum. However, the system at hand is bounded and thus it is not a continuum. Therefore, the RDF for all r bigger than the biggest possible distance in the system should be equal to zero; the RDF for this type of functions will be addressed as RDF*, indicating the discrete system. It should also be noted that the smallest value of r resulting in a non-zero value for the RDF* should be the closest distance possible between two atoms. Peaks in the value of the RDF can be expected; several distances will occur more often than the average density which is an indication on the structure of the system.

2.4.3 Calculation of the number of Hydrogen bonds

The structure, forging and breaking of hydrogen bonds is a hot topic for a long time [27–30]. In general, a 'donor' atom and an 'acceptor' atom are needed to forge a hydrogen bond. The 'donor' atom shares a covalently bonded hydrogen atom with the 'acceptor' over a maximum distance of approximately 3Å and a maximum `O-H-O` angle of 20° [31, 32]. To give an example, an oxygen atom O^{2-} is negatively charged which allows it to accept two bonds. Furthermore, the oxygen atom of a water molecule can also act as a 'donor' twice, since it has two hydrogens to share. The total number of hydrogen bonds in the system is thus calculated by comparing every individual, non hydrogen atom with all other atoms, to which it is not covalently bonded, in the system. If the distance and angle between the set of atoms satisfy the conditions and if one of the atoms can act as a 'donor' while the other can act as an 'acceptor', the hydrogen bond is counted. In the system at hand it is possible to have those bonds between the oxygen atoms of two water molecules (≈95% of the total number of bonds), between a water molecule and a molecule of the DNA (note that both the water molecule as the molecule in the DNA can act as a 'donor') or between two atoms of the DNA. The last type of bonds are the hydrogen bonds which keep the two chains together and the breaking of these bonds is an indication that the DNA is melting.

2.4.4 Matlab as post processor

Matlab is a powerful mathematical engine used to handle big amounts of data. NAMD produces 20 different types of 'energies' like the total energy of the system or temperature on a, by the user, specified frequency. When the data is saved just once in every 500 time steps of 2 fs this will still result in 20.000

data ponts for a 1 ns simulation. Putting this information in regular matrices (2D/3D) or cell arrays makes it possible to process this data in an efficient way. To do so, Matlab is used to read the NAMD output files and the useful data is saved in those types of matrices. Adding the results of a restarted simulation to the original simulation all the results are shown at once. The full Matlab script is shown in appendix D.17.

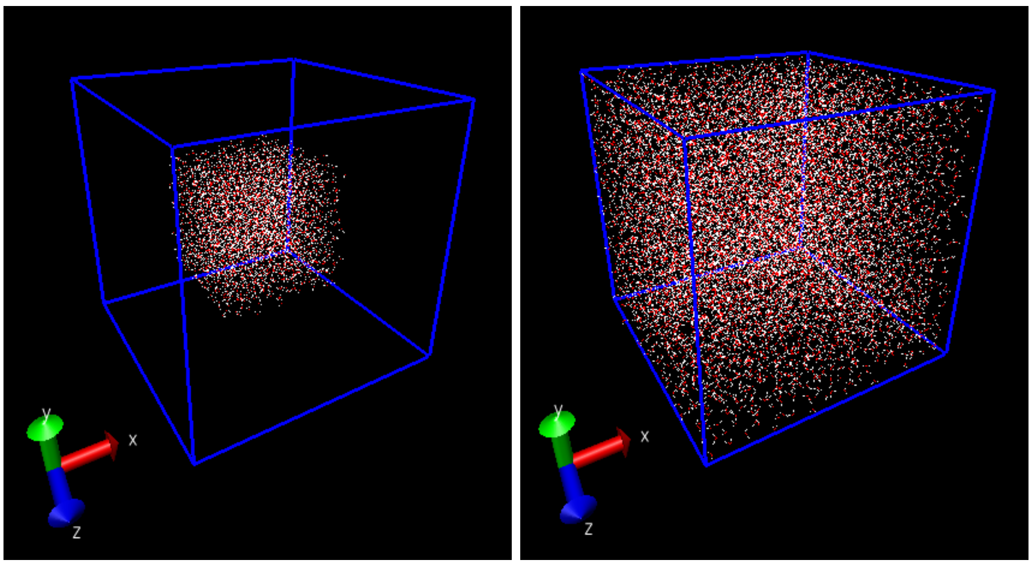
3 Model validation

The preliminary DNA simulations are solvated with the use of the TIP3P [33] water model. This model is a so-called three point model with rigid bonds and angles which reduces the computational complexity and thus time; all the bonded parts of the potential (equation 9) are zero for all time steps. The downside of reducing the computational complexity is the fact that the accuracy is also decreased. Therefore, the model is validated with the use of two systems (§3.1) by comparing calculated physical quantities (density in §3.2 and RDF in §3.3) with the same quantities but then gathered from experimental research.

3.1 Systems

Two systems are used to validate the model. Applying different boundary conditions to each of the systems provides insight in the validity of the structures and the applied software. Therefore, the first system is a cube of water with the dimensions $30 \times 30 \times 30\text{Å}^3$ which fits 5943 atoms which is equivalent to 1981 water molecules. The periodic boundary box for this first system is set to $60 \times 60 \times 60\text{Å}^3$, shown in figure 3.1(a). The second system (figure 3.1(b))is described by a box of water with the dimensions $60 \times 60 \times 60\text{Å}^3$ with the same initial periodic boundary box and containing 20535 atoms. Note that the first system is much smaller then the PBC which results in a PBC of constant dimensions while the second will stretch and/or squeeze the PBC due to the pressure in the system.

Both systems are heated up in a range from 150K up to 450K with the use of the default simulation parameters (see Appendix D.13). The temperature range covers the solid-fluid phase transition as well as the fluid-gas transition and the constant pressure control is applied to the periodic boundary box; the first system is thus a 'droplet' of water in vacuum under a certain pressure. The total simulation time (0.81ns) is relative small which implies that the heating rate is much bigger than the transition speed in nature. Therefore, it is not expected that both of the transitions are covered completely. The transitions are best seen in the first system since this system has more freedom to transform in shape; in the solid region it is staying at the original structure of a 'square' while it transforms to a more 'sphere-like' state in the fluid region which releases molecules in the gas region. This phenomenon is not seen in the second system since the size of the periodic boundary box is adapted to keep the current pressure constant; this bounds the system shape to the periodic boundary box.



(a) System 1 (b) System 2

Figure 3.1: The two systems

3.2 Density

Density is an important quantity when a phase transition is searched and a phase transition is a typical behavior which the numerical model should cover. As explained in the previous section, the first system changes shape for increasing temperature. Furthermore, the periodic boundary box is not equal to the 'droplet'. Measuring the volume of the water is thus very complicated. Therefore, a fixed part of the system is used to calculate the density. To do so, equation 31 is applied to this box, counting the number of oxygen atoms. This method is applied to both systems on a box of $20 \times 20 \times 20\text{Å}^3$ which is completely filled during the whole simulation. The gathered density data is averaged on each 50K and shown in a boxplot. From this plot, see figure 3.3, one can see that the behavior for both of the systems matches the general behavior of H_2O; there should be an increase in density when the system is heated and it should decrease after the solid-fluid transition. This behavior is present but does not occur around the real temperature of highest density which is $4°C$ [34].

3.3 Radial Distribution Function

As pointed out in §2.4.2, the RDF for a continuous system differs in several points from the discontinuous system. The RDF calculator, as provided by VMD, considers the systems at hand as discontinuous systems (even though PBC are applied) and the results are thus analyzed for the RDF*. To calculate the RDF* values, all oxygen atoms in both systems are selected and all the oxygen-oxygen distances are used to produce the RDF*. Therefore, the smallest expected distance is the oxygen-oxygen distance over a hydrogen bond, which is the length of the covalently oxygen-hydrogen bond plus the length of the hydrogen bond between the same hydrogen atom and a new oxygen atom (≈3Å). This peak (I) should occur independent of the phase of the system. These hydrogen bonds are forged by a summation of Lennard-Jones potentials acting between the two oxygens and between a oxygen-hydrogen (this pair is not covalently bonded) distance [25]. However, the phase of the systems determines the height of the peak; the more structure, the higher the peak. Therefore, it is expected that the RDF* plots for the solid states should return the highest peak while the gas states should return the lowest. Additionally, one can expect a drop (II) just after this peak due to the missing of oxygen atoms at distances slightly bigger then the hydrogen bond distance (with more molecules missing for the solid states). Furthermore, the so-called drift (which is the convergence to zero for increasing r) should start at smaller r in the systems at gas phase with respect to the systems at solid or fluid phases, but biggest r at which the RDF* returns a non-zero value should be bigger for the gas phase in comparison with the solid and fluid phased system: If there is a non-zero RDF* value at a big r it implies that this specific distance occurs at least once in the system. Knowing this, it can be stated that this value cannot exceed 1) the body diagonal in of the states where the system stays cubic, 2) the diameter of the sphere in the liquid state of the first system and 3) the body diagonal of the periodic boundary box for the gas states. Considering the solid and liquid states, one can imagine that there are several structural configurations present. For example, the solid state should be equivalent to 'ice ih' [34] which is hexagonal build with tetrahedral structures [35, 36]. Tetrahedral structures have, besides the hydrogen bonds forming them, also the $\approx \sqrt{2}$ times the length of the hydrogen bonds present which make the base of the tetrahedral. Therefore, a secondary peak (III) is expected at $3 \times \sqrt{2}\text{Å}$ which is wider then the primary peak at 3Å since the top angle of the tetrahedrals is not fixed at 90° [37]. Literature indicates that peaks (IV) and (V) for bigger r are expected due to higher order structures [26].

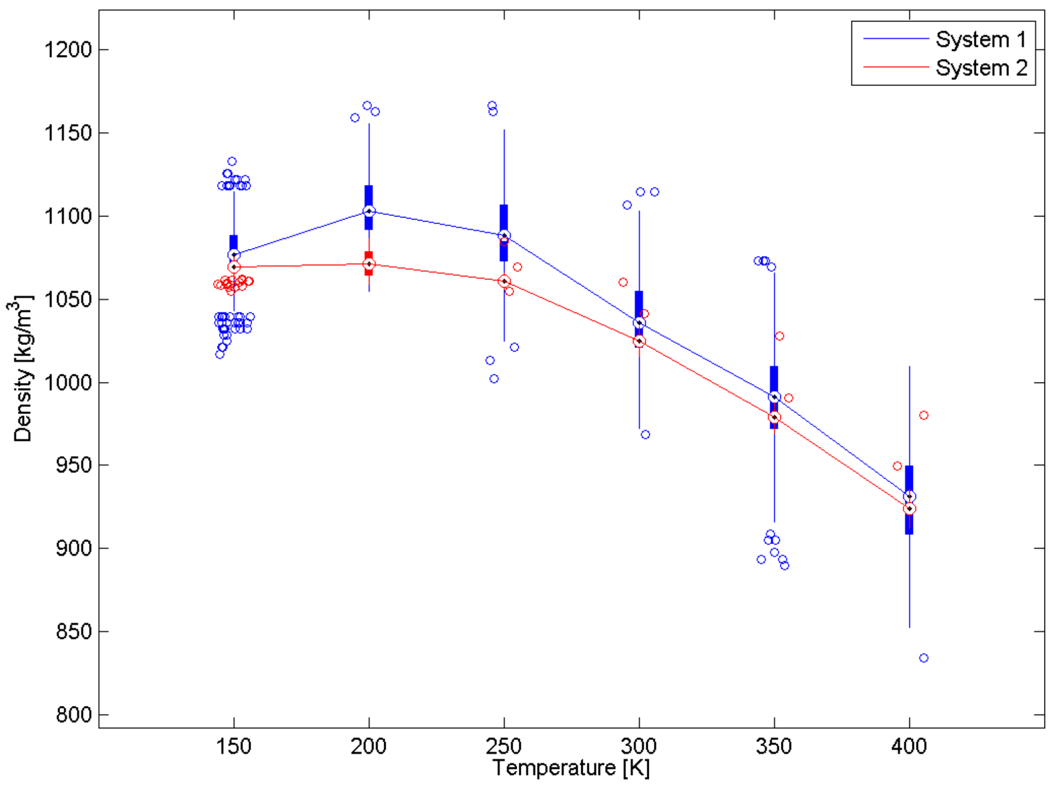


Figure 3.2: Density versus temperature plot for the two systems

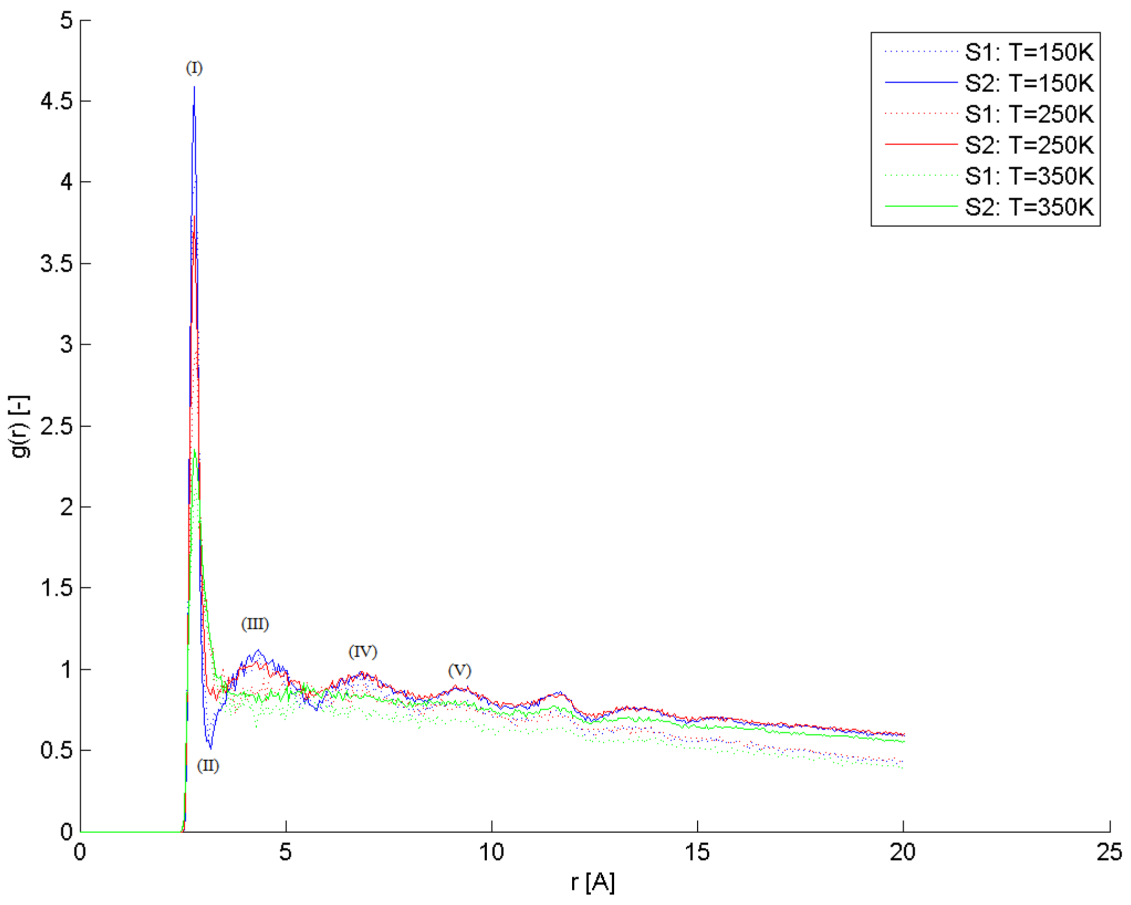
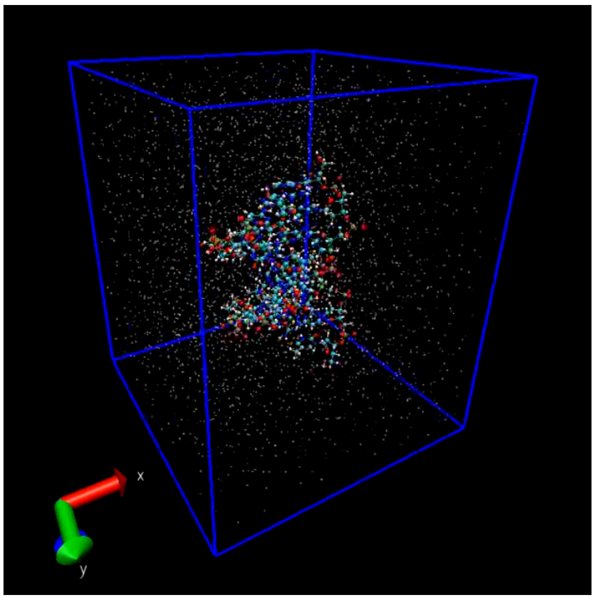


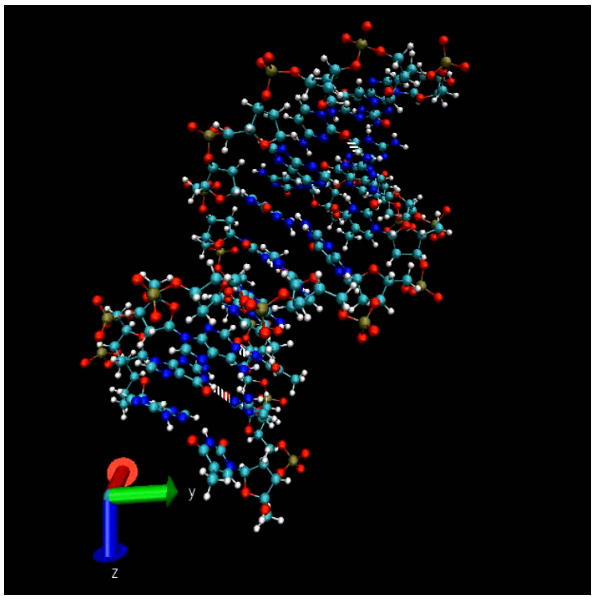
Figure 3.3: Discrete system radial distribution functions (RDF*) for the two discrete systems at three different temperatures

4 Results

A system consisting of the 10 base-pair λ - phage half-operator DNA surrounded by an ionized solvent is simulated, see figure 4.1(a). The system contains 9949 atoms from which 633 are related to the DNA, 70 to the ions (44 Na^+ and 26 Cl^-) and the rest to the water. A starting temperature of 300K is used since both, the water and the DNA, are equilibrated on this temperature. The system is prepared with 5.000 minimization and 100.000 equilibration steps (0.2ns) before it is heated with 0.1ns per 5K up to a temperature of 350K. On this temperature a final simulation of 1ns is performed and the results are gathered during the whole process. These results show a stable simulation; the DNA is not falling apart within the simulation which indicates that the unstable strongly negative charged chains are kept in place with the use of the solvent. The simulation also shows the initialization of 'melting'; the hydrogen bonds in the bottom of the chains (holding those two groups together) are broken at the end of the simulation. Nevertheless, it should be noted that it was a preliminary simulation with a high heating rate and a relative short total simulation time. Especially when it is compared to similar works in the field which are simulated for 250ns. The final configuration of the DNA is shown in figure 4.1(b).



(a) The system



(b) Final DNA structure

Figure 4.1: The two systems

5 Conclusion

Simulating DNA with H_2O, Na^+ and Cl^- molecules involves a lot of knowledge in several fields. In this study it was stated that the DNA is unstable due to the strongly negatively charged phosphate groups which tend to tear it apart. Applying water and additional ions to the system is done to neutralize the system and thus to stabilize the system. The simulations are performed with the use of several software packages which had to be understand by itself. This conclusion summarizes the structure of the DNA, the simulation approach and the validity of the model.

The DNA is build from two sugar-phosphate chains linked together by hydrogen bonds between the purine and pyrimidine groups which result in a double stranded structure discovered by Watson and Crick [5]. In experiments, DNA sequences of 30.000 base pairs [14] are used while simulations cannot reach such a big number of molecules. Therefore, studying this topic involves the search for bridges; a translation from the small simulation scale up to the big experiment scale. To do so, melting is one of the phenomena which can be used to investigate the flexibility of the DNA itself [10]. Knowing this flexibility is a step in understanding the behavior of the DNA in the bigger scales.

With the structure of the DNA at hand several software packages (VMD, NAMD and Matlab) are used to prepare, simulate and analyze the system. Applying the conjugate gradient method to the initial system results in a local minimum in total energy which implies that the simulations start at a semi-stable state. By performing equilibration straight after this minimization the lowest total energy of the system is found and the state is further stabilized. This equilibration is done on base of the Verlet algorithm which makes use of a scalar potential energy function to determine the forces. Applying a simple three point water model in the solvent reduces complexity of this potential (all the bonded parts are dropped out) but the accuracy is also reduced. The heating is performed on prescribing a new temperature by changing the parameters in a Langevin dynamics based temperature control. New equilibration steps are performed on each step and data is gathered during the whole process.

Having the simulations performed, the model is validated on base of the TIP3P water model. Investigating the density and the radial distribution functions of the system showed that this model behaves similar to H_2O in nature. Taking the computational time also into account makes clear that the results, at this stage, are acceptable. The DNA simulation showed that the system is well prepared and therefore stable. Applying a longer heating protocol can result in melting of this DNA.

6 Recommendations

An internship in a foreign country results in several recommendations which can, and perhaps should, be used in further research on the topic. This section describes the points of interest on the used software, the applied mathematical post-process calculations and the possibilities in extending the problem.

Using a commercial software package like NAMD as the simulation engine is similar in using a 'black box'; a commercial code which is simply used by hitting the right button. Therefore, it is hard to get insight in all the details for each step in the calculation. For example, the constant pressure control, which is based on the Langevin piston Nosé Hoover theorem, is used to control the pressure. Applying a 1atm pressure resulted in a 'noise' on this pressure of ± 100atm. By digging in the details for this specific part of the calculations resulted in an the discovery of an inconsistency between the user guide [23] and the theory [21, 22]. It is therefore strongly advised to do more simulations with the purpose of discovering the influence of these high fluctuations. A second important issue with the calculation details is the mathematical approach to hydrogen bonds; the literature [25] prescribe a fitted 12-6 Lennard Jones potential for these bonds while it is possible that the actual hydrogen bond calculation is done with a 10-12 Lennard Jones potential [38].

The mathematical post-process calculation of the RDF* is based on a discrete approach. However, it is possible to calculate the RDF value for the discrete system as well, this is done by taking a small part of the system (in general 1/3rd) which is compared to all other atoms in the system. In this way, the RDF value should converge to 1 on a cut-off distance r (which is 1/3rd of the system size). By recalculating, the results should match the experimental retrieved results [26] even better.

Expanding the problem can be done in many ways like, implementing longer DNA sequences, increasing the number of time steps or adding other and/or more ions. At the current stage, the programs written in this internship allows further research on these options in a fast and relative easy way. It is advised by S.A. Harris to increase the simulation time up to 250ns in which the melting should occur. Nevertheless, further research is needed on the simulation protocol (compare the current protocol with for example the protocol described by Shields *et al.* [12]), by getting more insight in the thermal coupling of the DNA and the solvent a more stable simulation is expected. Further research is also needed for the simulation temperature; melting occurs faster on higher temperatures. It is unknown if this temperature is bounded to a maximum. After the simulations for the short sample, the system should be expanded to see whether or not the length of the DNA is of influence on the melting temperature. It is also possible to continue the work of Harris, S.A. *et al.* to investigate the influence of mismatches in the sequence.

References

[1] William Humphrey, Andrew Dalke, and Klaus Schulten. VMD – Visual Molecular Dynamics. `http://www.ks.uiuc.edu/Research/vmd/`, 1996. Journal of Molecular Graphics, 14:33-38.

[2] James C. Phillips, Rosemary Braun, Wei Wang, James Gumbart, Emad Tajkhorshid, Elizabeth Villa, Christophe Chipot, Robert D. Skeel, Laxmikant Kale, and Klaus Schulten. Scalable molecular dynamics with NAMD. `http://www.ks.uiuc.edu/Research/namd/`, 2005. Journal of Computational Chemistry, 26:1781-1802.

[3] Tcl Core Team. TCL Developer Xchange. `http://www.tcl.tk`, 1988.

[4] J.D. Baleja, R.T. Pon, and B.D. Sykes. Solution structure of phage lambda half-operator DNA. `http://www.pdb.org/pdb/explore/explore.do?structureId=1D20`, 2005. Biochemistry 29:4828-4839.

[5] Scott Freeman. Nucleic Acids and the RNA World, 2005. Pearson, BIOLOGICAL SCIENCE third edition, 67-81.

[6] M.S. Smyth and J.H.J Martin. x Ray crystallography, 2000. J. Clin. Pathol: Mol Pathol 53:8-14.

[7] H. Long, A. Kudlay, and G.C. Schatz. Molecular Dynamics Studies of Ion Distributions for DNA Duplexes and DNA Clusters: Salt Effects and Connection to DNA Melting, 2006. Phys. Chem. B 110:2918-2926.

[8] M.M. Senior, R.A. Jones, and K.J. Breslauer. Influence of loop residues on the relative stabilities of DNA hairpin structures, 1988. Proc. Natl. Acad. Sci. USA 85:6242-6246.

[9] D. Sprous, M.A. Young, and D.L. Beveridge. Molecular Dynamics Studies of the Conformational Preferences of a DNA Double Helix in Water and an Ethanol/Water Mixture: Theoretical Considerations of the A $\leftrightarrow$ B Transition, 1998. J. Phys. Chem. B 102:4658-4667.

[10] G. Weber, J.W. Essex, and C.... Neylon. Probing the microscopic flexibility of DNA from melting temperatures, 2009. Nature Phys. 5:769-773.

[11] J.D. Baleja, R.T. Pon, and B.D. Sykes. Solution Structure of Phage λ Half-Operator DNA by Use of NMR, Restrained Molecular Dynamics, and NOE-Based Refinement, 1990. Biochemistry 29:4828-4839.

[12] G.C. Shields, C.A. Laughton, and M. Orozco. Molecular Dynamics Simulations of the d(T·A·T) Triple Helix, 1997. J. Am. Chem. Soc. 119:7463-7469.

[13] J.R. Shewchuk. An Introduction to the Conjugate Gradient Method without the agonizing pain, 1994. School of Computer Science, Carnegie Mellon University Pittsburgh, PA 15213.

[14] Lívia S. Gomes, Márcio S. Rocha, Henrique W. Moysés, Oscar N. Mesquita, Mônica C. de Oliveira, and Sônia M.L. de Silva. Study of DNA - Cyclodextrin interaction using optical tweezers and photon correlation spectroscopy, 2009.

[15] RCSB Protein Data Bank. PDB Protein Data Bank. `http://www.pdb.org`, 2011.

[16] M.P Allen and D.J Tildesley. Computer Simulation of Liquids, 1987. Oxford University Press, New York.

[17] Phillips J.C., Villa E., Isgo T, and Sotomayer M. Scalable molecular dynamics with NAMD. `http://www.ks.uiuc.edu/Training/Tutorials/namd`, 2005. Journal of Computational Chemistry, 26:1781-1802.

[18] S.A. Harris. Modelling the biomechanical properties of DNA using computer simulation, 2006. Phil. Trans. R. Soc. A 364:3319-3334.

[19] H.C. Urey and C.A. Bradley. The vibrations of pentatonic tetrahedral molecules, 1931. Phys. Rev. Vol. 38 11:1969-1978.

[20] D. Quigley. Constant pressure langevin dynamics: Theory and application to the study of phase behaviour in core-softened systems, 2005. University of York, Department of Physics.

[21] G.J Martyna, D.J. Tobias, and M.L Klein. Constant pressure molecular dynamics algorithms, 1994. J. Chem. Phys. Vol. 101 5:4177-4189.

[22] S.E. Feller, Y. Zhang, R.W. Pastor, and B.R. Brooks. Constant pressure molecular dynamics simulation: The Langevin piston method, 1995. J. Chem. Phys. Vol. 103 11:4613-4621.

[23] M. Bhandarkar, A. Bhatele, E. Bohm, R. Brunner, F. Buelens, C. Chipot, A. Dalke, A. Dixit, G. Fiorin, P. Freddolino, P. Grayson, J. Gullingsrud, A. Gursoy, D. Hardy, C. Harrison, J. Hnin, W. Humphrey, D. Hurwitz, N. Krawetz, S. Kumar, D. Kunzman, C. Lee, C. Mei, M. Nelson, J. Phillips, O. Sarood, A. Shinozaki, G. Zheng, and Zhu F. NAMD User's Guide. http://www.ks.uiuc.edu/Research/namd/2.7b2/ug/node23.html, 2009.

[24] Jr. MacKerell, A.D., B. Brooks, III Brooks, C. L., L. Nilsson, B. Roux, Y. Won, and M. Karplus. CHARMM: The Energy Function and Its Parameterization with an Overview of the Program. http://www.charmm.org/, 1998. Schleyer, P.v.R., et al.. The Encyclopedia of Computational Chemistry. 1. Chichester: John Wiley & Sons. pp. 271-277.

[25] N. Foloppe and A.D. MacKerell Jr. All-Atom Empirical Force Field for Nucleic Acids: 2. Parameter Optimization Based on Small Molecule and Condensed Phase Macromolecular Target Data. 2000. 21: 86-104.

[26] W.L. Jorgensen and J. Tirado-Rives. Potential energy functions for atomic-level simulations of water and organic and biomolecular systems, 2005. PNAS Vol. 102 19:6665-6670.

[27] G.C. Pimentel and A.L. McClellan. The hydrogen bond, 1960. Freeman, W.H. and company.

[28] G.A. Jeffrey. An introduction to hydrogen bonding, 1997. Oxford University Press.

[29] S.J Grabowski. Hydrogen Bonding: new insights, 2006. Springer.

[30] S. Scheiner. Hydrogen bonding: a theoretical perspective, 1997. Oxford University Press.

[31] W. Humphrey, A. Dalke, and K Schulten. VMD User's Guide. http://www.ks.uiuc.edu/Research/vmd/current/ug.pdf, 2011. Theoretical and Computational Biophysics Group, Beckman Institute for Advanced Science and Technology, University of Illinois at Urbana-Champaign.

[32] J.C. Gumbart. VMD - Hbonds plugin. http://www.ks.uiuc.edu/Research/vmd/plugins/hbonds/, 2011. Theoretical and Computational Biophysics Group, Beckman Institute for Advanced Science and Technology, University of Illinois at Urbana-Champaign.

[33] W. L. Jorgensen, J. Chandrasekhar, J. D. Madura, R. W. Impey, and M. L. Klein, 1983. J. Chem. Phys. 79, 926.

[34] F.H. Stillinger. Water revisited, 1980. SCIENCE, Vol. 209, 4455:451-457.

[35] P. Kumbar, S.V. Buldyrev, and H.E. Stanley. A tetrahedral entropy for water, 2009. PNAS Vol. 106 52:22130-22134.

[36] B. Chen, I. Ivanov, M.L. Klein, and M. Parrinello. Hydrogen bonding in water, 2003. Physical Review Letters, Vol. 91, 21:(215503)1-4.

[37] D.J. Price and C.L. Brooks. A modified tip3p water potential for simulation with ewald summation, 2004. J. of Chem. Phys. Vol. 121, 20:10096-10103.

[38] D.M. Ferguson and Kollman P.A. Can the lennardjones 6-12 function replace the 10-12 form in molecular mechanics calculations?, 1991.

[39] Persistence of Vision Raytracer Pty. Ltd. Povray. http://www.povray.org, 2008.

[40] D. van der Spoel, E. Lindahl, B. Hess, G. Groenhof, Mark A. E., and H. J. C. Berendsen. GROMACS: Fast, Flexible and Free. http://www.gromacs.org, 2005. J. Comp. Chem., 26:1701-1718.

[41] AMBER development team. Amber home page. http://www.ambermd.org, 2011.

[42] The Mathworks Inc. Matlab. http://www.mathworks.com/products/matlab/, 2011.

[43] Gnuplot. http://www.gnuplot.info, 2010.

[44] Grace development team.

Appendices

A Nomenclature

A.1 Normal symbols

Symbol	Unit	Description
b	kg/s^2	solution vector
d	Å	search direction
f	N	forces (MD)
k	-[1]	multiplicative constant
m	kg	mass
p	N/m^2	pressure
r	Å	position
s	Ns	momentum
t	-	time step

A.2 Capital symbols

Symbol	Unit	Description
A	kg/(Ås^2)	linear depence
N	-	number of atoms
R	-[1]	random force, random rate of change of volume
U	kcal/mol	potential
V	Å^3	volume
W	kg/m^4	relative piston mass

A.3 Sub- & superscripts

Symbol	Unit	Description
$_\theta$	-	angle
$_i$	-	direction in space
$_j$	-	iteration step
$_0$	-	initial
$_{14}$	-	electric scaling constant
$_m$	-	minimum
$_a$	-	index of primary particle
$_b$	-	index of secondary particle
$_c$	-	index of tertiary particle
$_d$	-	index of quaternary particle
$_{pq}$	-	in relation to the p^th and q^th particle
$_{ub}$	-	Urey-Bradley
^{T}	-	transpose operator

A.4 Greek symbols

Symbol	Unit	Description
α	-	weight function
β	-	Gram-Schmidt factor
δ	-	increment
ξ	-	periodicity factor
ψ	rad	angle
ϕ	rad	angle
ε	kcal/mol	potential depth
γ	1/s	friction constant
Σ	-	summation operator
∇	1/Å	gradient operator

[1]These variables are used in several situations and have different dimensions

A.5 Other symbols

Symbol	**Unit**	**Description**
$\dot{e}$	1/s	ratio in change of volume and volume
∂	-	partial derivative
$\mathcal{O}$	-	order
$\mathcal{R}$	Å	residual
$\mathcal{S}$	N	Added force in constant pressure control

B List of acronyms

Acronym	Description
A	Adenine
C	Cytosine
G	Guanine
T	Thymine
DNA	Deoxyribonucleic Acid
NpT	constant number of atoms, pressure and temperature ensemble
PDB	Protein Data Bank
PSF	Protein Structure File
RDF	Radial Distribution Function
RDF*	Radial Distribution Function for discrete systems
VMD	Visual Molecular Dynamics
NAMD	NAnoscale Molecular Dynamics
POVRAY	Persistence of Vision Raytracer
GROMACS	Groningen Machine for Chemical Simulations
CHARMM	Chemistry at Harvard Macromolecule Mechanics
TCL	Tool Command Language

C Software

Visualizing and simulating DNA is possible with different software packages. Examples of visualization packages are Visual Molecular Dynamics (VMD) [1] or Persistence of Vision Raytracer (POVRAY) [39] from which VMD is used for the simulations at hand, see §C.1. NAMD [2] is one of the several available software packages on the field of Molecular Dynamics (MD). Other available packages are for example: Groningen Machine for Chemical Simulations (GROMACS) [40], AMBER [41] or Chemistry at Harvard Macromolecule Mechanics (CHARMM) [24]. In this case, the NAMD software has been chosen, see §C.2.Concluding there are several data processors available to plot the results. Matlab [42], gnuplot [43] or xmgrace [44] are such programs from which Matlab is used, see §C.3.

C.1 VMD

VMD [1] is a very powerful visualization tool, but it is more. When VMD is used in combination with NAMD it is possible to use the TCL language [3] in combination with specialized VMD protocols to prepare, start and analyze NAMD simulations. Examples of such specialized protocols are the 'atomselect ' or 'measure ' commands which makes it possible to easy select any selection of the system at hand or measure, for example, the number of hydrogen bonds in the system. The language is capable of reading and writing files which makes it possible to prepare multiple systems and run them in the NAMD engine in a serial order and the generate new output which can be used by third-party programs like Matlab for further analysis of the results. Concluding, VMD has a extensive graphical user interface which, for example, allows users to load trajectory files and see animations of the simulated system.

C.2 NAMD

There are several reasons for using NAMD as the calculating engine. One of the mean reasons is the connection with VMD; since both of the programmes are developed by the same research group, they have a strong correlation in the in- and output. It is thus possible to use the VMD output straightforward into NAMD and vice versa. Another reason for choosing NAMD was due to the fact that NAMD is able to do parallelization over both the processors and the graphical cards in a computer. Furthermore, NAMD only requires one input file (.conf, see §2.3.7), it is well prepared for Steered Molecular Dynamics (SMD) [17] and it is free and open source.

C.3 Matlab

A logical choice for the plot tool is Matlab since this program is free available to students of the University of Twente. The software is also able to do much more than plotting; reading, understanding and writing all kinds of text files is an easy job and it is also well prepared for handeling tremendous amounts of data. Therefore, any other data manipulation is easily done in Matlab.

C.4 Interaction

Figure C.1 show the interaction between the three different software packages. More information on the individual steps is explained further in this report.

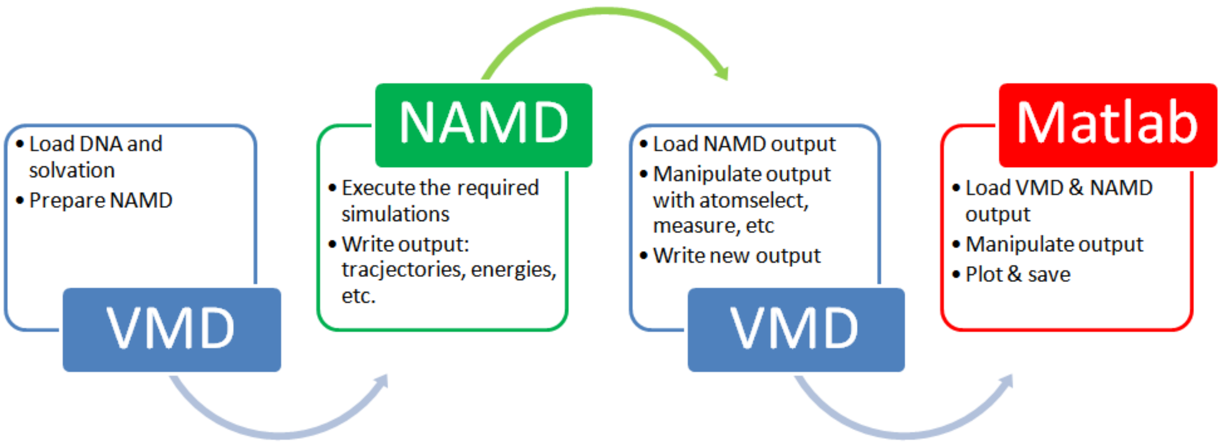


Figure C.1: Flowchart of software usage

D Scripts

This appendix contains all the scripts as used in the simulations. The order of the following sections is based on the level of the script; the script which is controls all others is the so-called ‘Flexible.tcl’ script in which all the others are called.

D.1 Flexible.tcl

```
#########################################################################
#This script should be the base for any kind of simulations, it should
#therefore be flexible!
#For now, it is not possible to implement DNA (yet)
##########################################################################



##########################################################################
#Initialisation of simulation specific parameters:
##########################################################################

#Set the number of systems. When nr_sizes is bigger then 1 the systemsizes
# will be according
#to boxsize=5+i*5, i=1,2,3.. . The standard boxsize for 1 system is 15.
#Don't try more then
#4 sizes.
set nr_sizes 1
if {$nr_sizes==1} {
        set boxsize 10
} else {
        set boxsize 10
        for {set i 2} {$i<=$nr_sizes} {incr i} {
                lappend boxsize [expr 5+$i*5]
        }
}

#### IMPORTANT: setting both restarts and temperature range is not possible!
#Set the number of restarts, default is 1 which implies: no restarts
set nr_rest 1

#Set the temperature range, the number of T's will thereafter be calculated.
set sT  0 ;#Additional switch for T-range calculations
if {$sT==1} {
        set Tmin 300    ;#Set starting temperature
        set T $Tmin
        set Tmax 350    ;#Set ending temeprature
        set dT 5        ;#Set delta T
        set nr_T [expr ($Tmax-$Tmin)/$dT]       ;#Calculate the nr of T steps
} else {
        set T 300
}

#Set the default time parameters
set mintime 500 ;#Number of minimization steps
set mintimeh $mintime ;#Save the initial number of minimization steps for
#the hbonds counting
set eqtime 1000 ;#Set the number of equilibration time steps.
set runtime 500 ;#Number of running steps, must be x*100
set OPfreq [expr $runtime/100] ;#The frequency of all output generation

#Load the switches for master.tcl default: 1 0 1 0 1 0 1 0 0 0 1 0
#More info in Switches.tcl
source Switches.tcl
```

```
#Load additional functions
source afrond.tcl       ;#This script is an improvement of the ceil, round
#functions
source Color_DNA.tcl    ;#This script can color the DNA and waterbox
source cylinder2.tcl    ;#This script is able to draw a cylinder around the
#system
source writeconf.tcl    ;#This script produces the .conf files
##############################################################################


##############################################################################
#Space to declare 'dummy' variables. These variables could be used to enter
#loops etc.
##############################################################################

set inputname 0

##############################################################################


##############################################################################
#Initialize the systems, so prepare the default configuration parameters
# for all sizes, restarts and temperatures. A loop over the watermodels
#and DNA should also be included here.
###############################################################################

#Start the loop over the system sizes
for {set i 0} {$i<$nr_sizes} {incr i} {
        #Update the suffix tot the current file.
        set suffix _s$i

        #Set the RESTart switch back to 0 (changes when a restart run has passed)
        set sREST 0

#Only one folder with results should be made, therefore the 'MaKeFoLder'
#switch is turned off.
        if {$i==1} {
                set sMKFL 0
        }

#Run the master script with current switches. On current default settings this
#implies that the file locations are loaded, a results folder is made.
#When there is DNA present, a lot of other steps can be taken (loading DNA,
#splitting DNA, writing psf etc)
        source master.tcl


#Start the loop over the number of restarts, can handle up to 99 restarts.
        for {set j 0} {$j<$nr_rest} {incr j} {
                #Change the suffixes to double numbered indices
                if {$j<10} {
                        set sj 0$j
                } else { set sj $j }

                #Add the restart suffix to the configurations files
                set CONFfile $CONFbase${suffix}_r${sj}

                #Change the restart switch when the first restart starts.
                if {$j>0} {
                        set sREST 1
                }
                #Load the default configuration parameters
                source defaultconf.tcl

                #Update the outputname
                set outputname $CONFbase${suffix}_r${sj}


############################################################################
#Change default parameters to simulation specific parameters. Note that the
#default parameters only use one Temperature. Therefore, it is changed
#from here. Furthermore, the loop over the
#temperatures should be added in the configurations file.
############################################################################

                #Update the output name in the .conf files
                unset OP
                set OP $outputname
                lappend OP  $OPfreq  $OPfreq $OPfreq $OPfreq $OPfreq

                if {$sREST==1} {
                set mintime 0
                set EX 0
                lappend EX $mintime $T $runtime $eqtime
                }

                #Add the Temperature loop to the conf file if needed.
                if {$sT==1} {

                        unset EX        ;#Unset the old Execution parameters
                        set EX 1        ;#Set the type to multi temperature
                        #Prepare the minimization run:
        lappend EX "minimize $mintime \nrun $mintime \n"
        #Write the for loop over the Temperatures. This is executed by NAMD
        lappend EX "run $eqtime \n \n for {set T $Tmin} {\$T <= $Tmax}
        {incr T $dT} {\nlangevinTemp \$T\nlangevinPistonTemp \$T\nreinitvels
                        \$T\nrun $runtime\n}"

                #Closure of the $sT if statement
                }


##############################################################################
#Write the configuration files. Each new restart or system gets a new .conf
# file, each new T is placed in the current restart/system .conf file.
##############################################################################

writeconf $FL_results $CONFfile $job_description $inputstructure $restartpar
                $PARfile $T $FF $IP $CTC $PBC $PME $CPC $OP $EXTRA $EX
        #Closure of the restart loop
        }

#Closure of the systems loop
}


###############################################################################
#All the preparations are done, NAMD is called by Exec_NAMD.tcl. This script
#counts the number of .conf files in the results folder and starts the
#simulations in serial order.
###############################################################################
```

```
source Exec_NAMD.tcl


################################################################################
#The following script writes a very small output file for matlab which
#includes the names of all the .conf files
################################################################################

source matlabinfo.tcl

################################################################################
#This script calculates the number of hydrogen bonds in the system.
################################################################################

source hydro.tcl
```

D.2 Switches.tcl

```
################################################################################
#Set the switches:
#System = only waterbox?
set sONLYWATER 1

#Execute? yes/no [1/0]
set sOPEN 0     ;#Open a PDB file?
set sMKFL 1     ;#Make a new results folder?
set sPSFG 1     ;#PSF gen
set sWATE 1     ;#Add waterbox?
set sAION 0     ;#Add ions?
set sREST 0     ;#Is it a simulation which is restarted?
set sDECO 1     ;#Prepare conf - default way
set sWRIT 0     ;#Write conf
set sNAMD 0     ;#Run NAMD
set sTRAJ 1     ;#Add trajectory

#Choose colormethod:
#[0] = No DNA to color
#[1] = Coloring by group type
#[2] = Coloring by segment
set cmethod 1

#Switch drawing a cilinder on/off [1/0]
set drawcil 0

#Switches for default parameters in defaultconf.tcl (0 = take default, 1 = use
#user specified (in defaultconf.tcl)
set FF 0
set IP 0
set CTC 0
set PME 0
set CPC 0
set EX 0


################################################################################
```

D.3 afrond.tcl

```
proc afrond {mode x p} {
#This protocol is able to round a number in three ways:
#'u' - up: so the number is rounded up to the next integer
#'d' - down: so the number is rounded down to the next integer
#'p' - precision: the number is rounded regular to the decimal specified by p

#Initiate a 'sum of a' variable, used to check if the input is a decimal number
set suma 0

        #Loop over the 'length'  of the input number. This length is the string
         length, so every digit is translated in a string character
        for {set i 0} {$i<[string length $x]} {incr i} {
                #Find the . in the string
                set a [string equal [string index $x $i] "."]
                #If $a==1, save the number of i, this is the index of the dot
                if {$a==1} {
                        set ai $i
                }
                #Increase the sum of a with the value of a
                incr suma $a
        }
        #If this sum returns 0 it implies there is no dot, then the last number
         of the string is used to round.
        if {$suma==0} {set ai [expr [string length $x]]}

        #The up mode: take the first part of the string and add 1 to this part
        if {[string equal $mode "u"]} {
                set y1 [string range $x 0 $ai-1]
                set y [expr $y1 + 1]
        #The down mode, just take the first part of the string
        } elseif {[string equal $mode "d"]} {
                set y [string range $x 0 $ai-1]
        #The precision mode, save the first part and the part up the the p'th
         decimal (this is the index of the dot + the required precision) of
         the string
        } elseif {[string equal $mode "p"]} {
                set y1 [string range $x 0 $ai]
                set y2 [expr $ai + $p]
                #When the number @p is bigger then 4 round it upwards,
                otherwise round it downwards
                if {[string index $x $y2]>4} {
                        set ln [expr [string index $x $y2] +1]
                } else {set ln [string index $x $y2]}
                set y3 [string range $x $ai+1 $y2-1]
                #Put it all together in a new string
                set y ${y1}${y3}${ln}
        #Just returning an error msg to the console if the specified mode is
        not supported
        } else {puts "This mode is not supported, try 'u'=up, 'd'=down,
        'p'=precision, NB: give 0 precicion for 'u' and 'd'"}

return $y

}
```

D.4 Color_DNA.tcl

```
proc Color_DNA {cmethod molid} {
##############################################################################
# This file is used to give the system a new representation. Two options are #
# included. The first colors the groups and adds a ribbon thrue the backbones#
# while the second method only colors the segments. Water is shown as dots in#
# both of the methods. Call this function by: Color_DNA cmethod molid        #
##############################################################################
```

```
################################################################################
# Preliminary work:
# Delete original drawing method (which is lines, colored by elements for all
# elements in the system.
mol delrep 0 $molid

################################################################################

################################################################################
#Colormethod 1
if {$cmethod == 1} {
#Add Backbone representation: Yellow Ribbon
mol color ColorID 4
mol rep NewRibbons 0.600000 6.000000 3.000000 0
mol selection {backbone}
mol addrep $molid

#Add Guanine representation, Green CPK
mol color ColorID 7
mol rep CPK 1.00000 0.500000 8.000000 6.000000
mol selection {resname DG or resname GUA}
mol addrep $molid

#Add Cystosine representation, Orange CPK
mol color ColorID 3
mol rep CPK 1.00000 0.500000 8.000000 6.000000
mol selection {resname DC or resname CYT}
mol addrep $molid

#Add Adenine representation, Iceblue CPK
mol color ColorID 15
mol rep CPK 1.00000 0.500000 8.000000 6.000000
mol selection {resname DA or resname ADE}
mol addrep $molid

#Add Thymine representation, Blue CPK
mol color ColorID 0
mol rep CPK 1.00000 0.500000 8.000000 6.000000
mol selection {resname DT or resname THY}
mol addrep $molid

#Add Hydrogen bonds: White, dashed lines
mol color ColorID 8
mol rep HBonds 3.5 20.0 4.0
mol selection {backbone or resname DT or resname DA or resname DG or resname
DC or resname GUA or resname CYT or resname ADE or resname THY}
mol addrep $molid

} else {
################################################################################
#Colormethod 2

#Add segA representation, Blue CPK
mol color ColorID 0
mol rep CPK 1.00000 0.500000 8.000000 6.000000
mol selection {chain A or segname DNA1 and noh}
mol addrep $molid

#Add segB, Blue CPK
mol color ColorID 1
mol rep CPK 1.00000 0.500000 8.000000 6.000000
mol selection {chain B or segname DNA2 and noh}
mol addrep $molid

}
#Show the water
mol color Element
mol rep Points 1
mol selection {water}
mol addrep $molid
}
```

D.5 cylinder2.tcl

```
proc cylinder2 {molid xmin xmax ymin ymax zmin zmax} {
################################################################################
# By de-commenting the lines below it is possible to draw a cilinder around
# the DNA.
#Variables available to draw a cylinder:
set xavg [expr ($xmin + $xmax) / 2]
set yavg [expr ($ymin + $ymax) / 2]

set p1 $xavg
lappend p1 $yavg
lappend p1 $zmin
set p2 $xavg
lappend p2 $yavg
lappend p2 $zmax

set r 13.005
set n 10

graphics $molid cylinder $p1 $p2 radius $r resolution $n filled off

################################################################################
```

D.6 writeconf.tcl

```
proc writeconf {FL_results CONFfile job_description inputstructure restartpar
 PARfile T FF IP CTC PBC PME CPC OP EXTRA EX} {
#This protocol reads the information as specified by Flexible.tcl and
defaultconf.tcl and uses this to write a .conf file for every simulation.
set fileid [open ${FL_results}$CONFfile.conf w]

puts $fileid "
#############################################################
## JOB DESCRIPTION                                         ##
#############################################################

# $job_description

#############################################################
## ADJUSTABLE PARAMETERS                                   ##
#############################################################

structure          $inputstructure.psf





"
set inputname      [lindex $restartpar 2]
# only need to edit this in one place!
```

```
if {[lindex $restartpar 0]==1 } {
puts $fileid "#RESTART PARAMETERS HERE:


coordinates     $inputname.restart.coor  ;# coordinates
from last run (binary)
velocities      $inputname.restart.vel   ;# velocities
from last run (binary)
extendedSystem      $inputname.restart.xsc   ;# cell
dimensions from last run
#temperature         $T
firsttimestep      [lindex $restartpar 1]
# last step of previous run
numsteps           [lindex $restartpar 3]
# run stops when this step is reached

"
} else {
puts $fileid "
coordinates        $inputstructure.pdb
firsttimestep 0
temperature         $T
"
}

puts $fileid "

#############################################################
## SIMULATION PARAMETERS                                   ##
#############################################################

# Input
#margin              2.5
paraTypeCharmm      on
parameters          ../$PARfile.inp


"
if {[lindex $FF 0]==1} {
#Put user specified FF parameters
puts $fileid "
# Force-Field Parameters -user
exclude             [lindex $FF 1]
1-4scaling          [lindex $FF 2]
cutoff              [lindex $FF 3]
switching           [lindex $FF 4]
switchdist          [lindex $FF 5]
pairlistdist        [lindex $FF 6]
"
} else {
#Put the default FF parameters:
puts $fileid "
# Force-Field Parameters -default
exclude             scaled1-4
1-4scaling          1.0
cutoff              12.0
switching           on
switchdist          10.0
pairlistdist        13.5
"
```

```
}

if {[lindex $IP 0]==1} {
#Put user specified IP
puts $fileid "
# Integrator Parameters -user
timestep            [lindex $IP 1]
rigidBonds          [lindex $IP 2]
nonbondedFreq       [lindex $IP 3]
fullElectFrequency  [lindex $IP 4]
stepspercycle       [lindex $IP 5]
"
} else {
puts $fileid "
# Integrator Parameters -default
timestep            2.0  ;# 2fs/step
rigidBonds          all  ;# needed for 2fs steps
nonbondedFreq       1
fullElectFrequency  2
stepspercycle       10
"
}

if {[lindex $CTC 0]==1} {
puts $fileid "
# Constant Temperature Control -user
langevin            [lindex $CTC 1]
langevinDamping     [lindex $CTC 2]
langevinTemp        [lindex $CTC 3]
langevinHydrogen    [lindex $CTC 4]
"
} else {
puts $fileid "
# Constant Temperature Control -default
langevin            on    ;# do langevin dynamics
langevinDamping     5     ;# damping coefficient
(gamma) of 5/ps
langevinTemp        $T
langevinHydrogen    off    ;# don't couple langevin
bath to hydrogens
"
}

if {[lindex $restartpar 0]==0} {
puts $fileid "
# Periodic Boundary Conditions
cellBasisVector1    [lindex $PBC 0]    0.0   0.0
cellBasisVector2    0.0   [lindex $PBC 1]   0.0
cellBasisVector3    0.0    0.0   [lindex $PBC 2]
cellOrigin          [lindex $PBC 5] [lindex $PBC 4]
[lindex $PBC 5]
wrapAll             [lindex $PBC 6]


# PME (for full-system periodic electrostatics)
PME                 [lindex $PME 0]
PMEGridSpacing      [lindex $PME 1]
"
if {[lindex $PME 0]==0} {
puts $fileid "
#manual grid definition
PMEGridSizeX        [lindex $PME 2]
PMEGridSizeY        [lindex $PME 3]
PMEGridSizeZ        [lindex $PME 4]
"
}
}
# Constant Pressure Control:
if {[lindex $CPC 0]==1} {
puts $fileid "
# Constant Pressure Control (variable volume) -user
useGroupPressure      [lindex $CPC 1]
useFlexibleCell       [lindex $CPC 2]
useConstantArea       [lindex $CPC 3]

langevinPiston        [lindex $CPC 4]
langevinPistonTarget  [lindex $CPC 5]
langevinPistonPeriod  [lindex $CPC 6]
langevinPistonDecay   [lindex $CPC 7]
langevinPistonTemp    [lindex $CPC 8]
"
} else {
puts $fileid "
# Constant Pressure Control (variable volume)
-default[lindex $OP 0]
useGroupPressure      yes ;# needed for rigidBonds
useFlexibleCell       no
useConstantArea       no

langevinPiston        on
langevinPistonTarget  1.01325 ;#  in bar -> 1 atm
langevinPistonPeriod  100.0
langevinPistonDecay   50.0
langevinPistonTemp    $T
"
}

# Output
puts $fileid "
binaryrestart           no
outputName          [lindex $OP 0]

restartfreq         [lindex $OP 1]
dcdfreq             [lindex $OP 2]
xstFreq             [lindex $OP 3]
outputEnergies      [lindex $OP 4]
outputPressure      [lindex $OP 5]


#############################################################
## EXTRA PARAMETERS                                        ##
#############################################################

$EXTRA

#############################################################
## EXECUTION SCRIPT                                        ##
#############################################################

# Minimization
"
if {[lindex $EX 0]==0} {
```

```
puts $fileid "
minimize            [lindex $EX 1]
reinitvels          [lindex $EX 2]

run [lindex $EX 4]
"
} else {
for {set i 1} {$i<=[llength $EX]} {incr i} {
puts $fileid "
[lindex $EX $i]
"
}
}


close $fileid

}
```

D.7 master.tcl

```
###############################################################################
#This 'master.tcl' script is the base for every simulation. Combining this   #
#script with Switches.tcl and File_locations.tcl prepares the simulations.   #
#By calling this script with new switches makes it possible to do other steps#
#of the simulation.                                                          #
###############################################################################

###############################################################################
#Clean up any old information:
mol delete all          ;# Delete all old molecules
#clear                  ;# Clear the TK console console, this command is note
                        #supported by the text version of VMD and is therefore
                        #canceled out.
###############################################################################


###############################################################################
#Do some initialisation steps:
source File_locations.tcl               ;# Run the script: File_locations.tcl
if {$sONLYWATER==0} {
mol new ${DNAfile}.pdb                  ;# Open the original pdb file
set molnum [expr [molinfo num] -1]      ;# Retrieve the number of last mol
set molid [molinfo index $molnum]       ;# Save the index of this number

###############################################################################

###############################################################################
#Write to the console that it is starting with reading the initial conditions:
puts "Starting to read initial conditions"
puts "###################################"
puts "   "
#Run Ini.tcl
source Ini.tcl
#Write to the console that the reading has finished:
puts "Initial conditions are read"
puts "###################################"
puts "   "
###############################################################################

###############################################################################
#Write to the console that the psfgeneration started:
```

```
if {$sPSFG==1} {
puts "Starting PSF generation"
puts "###################################"
puts "   "
#Create psf file
source DNA_psf.pgn
#Write to the console that the psfgeneration has finished:
puts "PSF generation has finished"
puts "###################################"
puts "   "
}
##############################################################################
}
##############################################################################
#Write to the console that the waterbox is being created:
if {$sWATE==1} {
puts "Starting to make the waterbox"
puts "###################################"
puts "   "
#Create waterbox
source Waterbox.tcl
#Write to the console that the waterbox is made:
puts "The waterbox is made"
puts "###################################"
puts "   "
}
##############################################################################

##############################################################################
#Write to the console that the waterbox is being created:
if {$sAION==1} {
puts "Starting to neutralize the system"
puts "###################################"
puts "   "
#Create waterbox
source addion.tcl
#Write to the console that the waterbox is made:
puts "The system is neutralized"
puts "###################################"
puts "   "
}
##############################################################################

##############################################################################
#Write to the console that it is starting with preparing the
configurations files:
if {$sDECO==1} {
puts "Starting to prepare configurations files"
puts "###################################"
puts "   "
#Run defaultconf.tcl
source defaultconf.tcl
#Write to the console that it is finished with preparing:
puts "Finished preparing .conf's"
puts "###################################"
puts "   "
}
##############################################################################

##############################################################################
#Write to the console that it is starting with writing the configurations
```

```
files:
if {$sWRIT==1} {
puts "Starting to write configurations files"
puts "####################################"
puts "   "
#Run Minimization_protocol.tcl
writeconf $FL_results $CONFfile $job_description $inputstructure $restartpar
$PARfile $T $FF $IP $CTC $PBC $PME $CPC $OP $EXTRA $EX
#Write to the console that it is finished with writing:
puts "Finished writing .conf's"
puts "####################################"
puts "   "
}
###############################################################################

###############################################################################
#Write to the console that NAMD is started for minimization and equilibration:
if {$sNAMD==1} {
puts "Starting NAMD for minimization and equilibration"
puts "####################################"
puts "   "
#Minimize the system:
source Exec_NAMD.tcl
#Write to the console that NAMD has finised:
puts "System is minimized and equilibrated"
puts "####################################"
puts "   "
}
###############################################################################

###############################################################################
#Write to the console that it is starting with the rendering:
puts "Starting rendering"
puts "####################################"
puts "   "
#Run Color_DNA.tcl
Color_DNA $cmethod $molid
#Write to the console that the rendering has finished:
puts "Rendering is finished"
puts "####################################"
puts "   "
###############################################################################
```

D.8 File_locations.tcl

```
###############################################################################
#Information about all input files:                                           #
###############################################################################

#All script and data files should be in the same map:
#Base name of the original file         ;# Needed for Open.tcl
set DNAfile 1D20

#Base name of the topology file
set TOPOfile top_all27_prot_na          ;# Needed for DNA_psf.pgn
set PARfile par_all27_prot_na           ;# Needed for writeconf.tcl & NAMD
simulation

#NAMD path:                             ;# Needed for NAMD simulation
#Uncomment the right path or add your own.
#set FL_namd "/usr/local/NAMD_2.7_Linux-x86/"           ;#RHAF
#set FL_namd "/usr/local/NAMD_2.7_Linux-x86_64/"        ;#PLA
```

```
set FL_namd "~/NAMD_2.7_Linux-x86_64/"                  ;#Caprica

#Set name for conf. file                ;# Needed for writeconf.tcl
set waterbase waterbox
if {$sONLYWATER==1} {
        set CONFbase $waterbase
} else {
        set CONFbase DNA_${DNAfile}
}
set CONFfile $CONFbase$suffix


###############################################################################
#Information about all output files:                                          #
###############################################################################

#Set name for results folder:           ;# Needed to save all output in
the same directory
#If it is a new simulation a new folder is made, when it was a restart
simulation, the old folder is used.
set systemTime [clock seconds]
if {$sMKFL==1} {
set FL_results "results_[clock format $systemTime -format %y%m%d%H%M]/"
file mkdir $FL_results
} else {
set FL_results [lindex [lsort [glob results*/]] end]
}


#Set some names:

set watername $waterbase$suffix                 ;# Output from Waterbox.tcl
set DNAfilenew ${DNAfile}_new$suffix            ;# Output from DNA_psf.pgn
set outputname $CONFbase$suffix              ;# Outputname for writeconf.tcl
```

D.9 Ini.tcl

```
###############################################################################
#This TCL script returns all initial conditions for the DNA simulations.     #
###############################################################################

###############################################################################
#Specify some initial conditions
#set vel 0.0005         ;# Stretch velocity (A/fs)
#set t 10000000                 ;# Total time (fs)
#set dt 2               ;# Timestep (fs)
set dz 0                ;# Elongation (A)
                        ;# - has to be sorted out: set dz or set vel?:
#set temperature 300    ;# Temperature
set boxsize 10          ;# The size, in all directions for the waterbox (A)
###############################################################################

###############################################################################
#Write the DNA into 2 segments: (Needed for DNA_psf.pgn)
#(The segments (don't) exclude oxigen and hydrogen, since these atoms will be
# created again in the psfgenerator. Reason: Unknown)
set segA [atomselect $molid "chain A"]
$segA writepdb ${FL_results}DNA1.pdb
set segB [atomselect $molid "chain B"]
$segB writepdb ${FL_results}DNA2.pdb
###############################################################################
```

```
##############################################################################
#Calculate some dimensions: needed for  Waterbox.tcl & writeconf.tcl
set minmaxdim [measure minmax [atomselect $molid all]]
set xmin [lindex [lindex $minmaxdim 0] 0]
set xmax [lindex [lindex $minmaxdim 1] 0]
set ymin [lindex [lindex $minmaxdim 0] 1]
set ymax [lindex [lindex $minmaxdim 1] 1]
set zmin [lindex [lindex $minmaxdim 0] 2]
set zmax [lindex [lindex $minmaxdim 1] 2]

#Set new box size in z direction:
set nbs [expr ($dz + $boxsize)]

set cBV1 [expr ceil($xmax - $xmin) + 2 * $boxsize]
set cBV2 [expr ceil($ymax - $ymin) + 2 * $boxsize]
set cBV3 [expr ceil($zmax - $zmin + $dz + 2 * $boxsize)] ;#add new box size

set centr_x [expr ($xmax + $xmin)/2]
set centr_y [expr ($ymax + $ymin)/2]
set centr_z [expr $cBV3/2 - $boxsize + $zmin]
##############################################################################

#Draw the cilinder if needed:
if {$drawcil==1} {
cylinder2 $molid $drawcil $xmin $xmax $ymin $ymax $zmin $zmax
}
```

D.10 DNA_psf.pgn

```
#More info about psf generation in:
#J. Gullingsrud, J. Phillips, psfgen User's Guide, version 1.0,
#April 10, 2002

#Open the psfgen generator, without this the psf can't be made
package require psfgen

#The generator is reset by this command, just to clean up
resetpsf

#Then a topology file is needed
#More info: http://www.ks.uiuc.edu/Training/Tutorials/science/
#forcefield-tutorial/forcefield-html/node6.html
topology    ${TOPOfile}.rtf

#Since there might be (and are) name inconsistancies in atoms
#or residues between the $DNAfile and the $TOPOfile, aliases
#are made.
#These are coppied from the autopsf function.
pdbalias residue DG   GUA
pdbalias residue DC   CYT
pdbalias residue DA   ADE
pdbalias residue DT   THY
pdbalias residue DU   URA
pdbalias  name GUA   O5*   O5'
pdbalias  name GUA   C5*   C5'
pdbalias  name GUA   O4*   O4'
pdbalias  name GUA   C4*   C4'
pdbalias  name GUA   C3*   C3'
pdbalias  name GUA   O3*   O3'
pdbalias  name GUA   C2*   C2'
pdbalias  name GUA   O2*   O2'
pdbalias  name GUA   C1*   C1'
pdbalias  name GUA   OP1   O1P
pdbalias  name GUA   OP2   O2P
pdbalias  name CYT   O5*   O5'
pdbalias  name CYT   C5*   C5'
pdbalias  name CYT   O4*   O4'
pdbalias  name CYT   C4*   C4'
pdbalias  name CYT   C3*   C3'
pdbalias  name CYT   O3*   O3'
pdbalias  name CYT   C2*   C2'
pdbalias  name CYT   O2*   O2'
pdbalias  name CYT   C1*   C1'
pdbalias  name CYT   OP1   O1P
pdbalias  name CYT   OP2   O2P
pdbalias  name ADE   O5*   O5'
pdbalias  name ADE   C5*   C5'
pdbalias  name ADE   O4*   O4'
pdbalias  name ADE   C4*   C4'
pdbalias  name ADE   C3*   C3'
pdbalias  name ADE   O3*   O3'
pdbalias  name ADE   C2*   C2'
pdbalias  name ADE   O2*   O2'
pdbalias  name ADE   C1*   C1'
pdbalias  name ADE   OP1   O1P
pdbalias  name ADE   OP2   O2P
pdbalias  name THY   O5*   O5'
pdbalias  name THY   C5*   C5'
pdbalias  name THY   O4*   O4'
pdbalias  name ADE   C4*   C4'
pdbalias  name ADE   C3*   C3'
pdbalias  name ADE   O3*   O3'
pdbalias  name ADE   C2*   C2'
pdbalias  name ADE   O2*   O2'
pdbalias  name ADE   C1*   C1'
pdbalias  name ADE   OP1   O1P
pdbalias  name ADE   OP2   O2P
pdbalias  name THY   O5*   O5'
pdbalias  name THY   C5*   C5'
pdbalias  name THY   O4*   O4'
pdbalias  name THY   C4*   C4'
pdbalias  name THY   C3*   C3'
pdbalias  name THY   O3*   O3'
pdbalias  name THY   C2*   C2'
pdbalias  name THY   O2*   O2'
pdbalias  name THY   C1*   C1'
pdbalias  name THY   OP1   O1P
pdbalias  name THY   OP2   O2P
pdbalias  name URA   O5*   O5'
pdbalias  name URA   C5*   C5'
pdbalias  name URA   O4*   O4'
pdbalias  name URA   C4*   C4'
pdbalias  name URA   C3*   C3'
pdbalias  name URA   O3*   O3'
pdbalias  name URA   C2*   C2'
pdbalias  name URA   O2*   O2'
pdbalias  name URA   C1*   C1'
pdbalias  name URA   OP1   O1P
pdbalias  name URA   OP2   O2P
pdbalias  name ILE   CD1   CD
pdbalias  name SER   HG   HG1
pdbalias  residue HIS   HSD
pdbalias  residue HEM   HEME
pdbalias  name HEME   N A   NA
pdbalias  name HEME   N B   NB
pdbalias  name HEME   N C   NC
pdbalias  name HEME   N D   ND
pdbalias  residue HOH   TIP3
pdbalias  name TIP3   O   OH2
pdbalias  residue K   POT
pdbalias  name K   K   POT
pdbalias  residue ICL   CLA
pdbalias  name ICL   CL   CLA
pdbalias  residue INA   SOD
pdbalias  name INA   NA   SOD
pdbalias  residue CA   CAL
pdbalias  name CA   CA   CAL
pdbalias  residue ZN   ZN2
pdbalias  name LYS   1HZ   HZ1
pdbalias  name LYS   2HZ   HZ2
pdbalias  name LYS   3HZ   HZ3
pdbalias  name ARG   1HH1   HH11
pdbalias  name ARG   2HH1   HH12
pdbalias  name ARG   1HH2   HH21
pdbalias  name ARG   2HH2   HH22
pdbalias  name ASN   1HD2   HD21
pdbalias  name ASN   2HD2   HD22


#Build the first segment (the first group is noted as the 5'
#part of the DNA and the last group is noted as the 3' part)
segment DNA1 {
first 5TER
last 3TER
pdb ${FL_results}DNA1.pdb
}

#Since the numbering in the original file goes: up-down-down-up,
#the first group in the second segment is the 5' part and the
#last is the 3' part.
segment DNA2 {
first 5TER
last 3TER
pdb ${FL_results}DNA2.pdb
}

#Apply a patch to one or more residues. Patches make small
#modifications to the structure of residues such as converting
#one to a terminus, changing the protonation state, or creating
#disulphide bonds between a pair of residues.
#(Text from user's guide)
patch DEO1 DNA1:1
patch DEO2 DNA1:2
patch DEO1 DNA1:3
patch DEO1 DNA1:4
patch DEO1 DNA1:5
patch DEO2 DNA1:6
patch DEO1 DNA1:7
patch DEO2 DNA1:8
patch DEO2 DNA1:9
patch DEO2 DNA1:10
patch DEO2 DNA2:11
patch DEO2 DNA2:12
patch DEO2 DNA2:13
patch DEO1 DNA2:14
patch DEO2 DNA2:15
patch DEO1 DNA2:16
patch DEO1 DNA2:17
patch DEO1 DNA2:18
patch DEO2 DNA2:19
patch DEO1 DNA2:20

#This line regenerates angles and dihedrals,
#When this is left out a error message in the NAMD
#simulation will occur
#Solution for this error was found in:
#http://ftp.ks.uiuc.edu/Research/namd/mailing_list/namd-l/9016.html
regenerate angles dihedrals

#Read coordinates from PDB file, matching segment,
#residue and atom names.
coordpdb ${FL_results}DNA1.pdb DNA1
coordpdb ${FL_results}DNA2.pdb DNA2

#Guesses coordinates of atoms for which they were not
#explicitly set. Calculation is based on internal coordinate
#hints contained in toplogy definition files. When these are
#insufficient, wild guesses are attempted based on bond
#lengths of 1   and angles of 109 .
#(Text from user's guide)
guesscoord

#Write out all information in new psf and pdb files.
writepsf ${FL_results}${DNAfilenew}.psf
writepdb ${FL_results}${DNAfilenew}.pdb
```

D.11 Waterbox.tcl

```
###############################################################################
#     This TCL script builds a square waterbox around the DNA molecule       #
###############################################################################

#Load the solvate package
#More info on:
#http://www.ks.uiuc.edu/Research/vmd/plugins/solvate/
package require solvate

if {$sONLYWATER==0} {
#Run the solvate parameters with the following options:
#  Use the new DNA files (both psf and pdb) as made by the DNA_psf.pgn
#  as base for the solvation
#- minmax: to specify the min. and max. dimensions of the DNA. These
#  are calculated in Ini.tcl, the waterbox takes this size now
#- t: add the $boxsize dimensions to all the directions of minmax
#- o: Write the output to files with this basename
#+ z: Elongate the system in z direction even more. Note that this
#     dimension cancels the -t in +z direction!
solvate ${FL_results}${DNAfilenew}.psf ${FL_results}${DNAfilenew}.pdb
-minmax $minmaxdim -t [lindex $boxsize $i] -o ${FL_results}$watername +z $nbs

#Since this protocol adds a new molecule, the old one should be deleted:
mol delete $molid
} else {
solvate -t [lindex $boxsize $i] -minmax {{0 0 0} {0 0 0}} -o
${FL_results}$watername
}
```

```
#Retrieve the ID of the added molecule (and overwrite the old
#variable molid.
set molnum [expr [molinfo num] -1]
set molid [molinfo index $molnum]

set boxdim [measure minmax [atomselect $molid all]]
```

D.12 addion.tcl

```
#Load the ionize package
package require autoionize

set ionpref ion
set ionconc 0.0
autoionize -psf ${FL_results}$watername.psf -pdb
${FL_results}$watername.pdb -is $ionconc -o ${FL_results}${ionpref}_$watername
mol delete $molid

#Retrieve the ID of the added molecule (and overwrite the old
#variable molid.
set molnum [expr [molinfo num] -1]
set molid [molinfo index $molnum]

set nametemp $waterbase
unset waterbase
set waterbase ${ionpref}_${nametemp}
set watername $waterbase$suffix
```

D.13 defaultconf.tcl

```
################################################################################
#Phase 0:
set job_description "Running $nr_sizes systems with the boxsizes of
$boxsize for $runtime ts with $mintime minimization.
#If hydro.tcl failed, redo hydro.tcl on your own computer with:

#nr_sizes = $nr_sizes
#nr_rest = $nr_rest
#FL_results = $FL_results
#CONFbase = $CONFbase
#waterbase = $waterbase
#mintimeh = $mintimeh
#eqtime = $eqtime
#OPfreq = $OPfreq
#and source afrond.tcl
"

if {$sONLYWATER==1} {
set cBV1 [expr 2 * [lindex $boxsize $i]]
set cBV2 $cBV1
set cBV3 $cBV1
set centr_x [lindex $boxsize $i]
set centr_y [lindex $boxsize $i]
set centr_z [lindex $boxsize $i]
}

set restartpar $sREST
#Add aditional information to restartpar if $sREST==1
if {$sREST==1} {
lappend restartpar [expr [lindex $EX 3] + ($j -1)*$runtime]
$outputname  [expr [lindex $EX 3]    + $j*$runtime]
set inputname      [lindex $restartpar 2]
```

```
# only need to edit this in one place!
}

set inputstructure $watername

#Force field parameters

#Add aditional information to FF if FF(0)==1
if {[lindex $FF 0]==1} {
lappend FF scaled1-4 1.0 5 on 5 5
#Exclude 1-4scaling cutoff switching switchdist pairlistdist
}

#Integrator parameters

#Add aditional information to FF if FF(0)==1
if {[lindex $IP 0]==1} {
lappend IP 2.0 all 1 2 10
#Time step, rigidBonds, nonbondedFreq, fullElectFrequency, stepspercycle
}

#Constant Temperature Parameters

if {[lindex $CTC 0]==1} {
lappend CTC on 10 $T on
#switch langevin, LDamping, LTemp, LHydrogen
}

#Set Periodic Boundary Conditions (always user defined)
set PBC $cBV1
lappend PBC $cBV2 $cBV3 $centr_x $centr_y $centr_z on
#Cell base vectors 1, 2 & 3, x,y,z coordinates and wrapAll (on/off)

#Set PME configurations (only apply manual grid size if PME

lappend PME 1
if {[lindex $PME 0]==0} {
lappend PME 45 45 48
#Set manual PME grid definition (x, y, z directions)
}

#Set Constant Pressure Control parameters:

if {[lindex $CPC 0]==1} {
lappend CPC yes no no on 1.01325 100.0 50.0 $T
#useGroupGressure, useFlexibleCell, useConstantArea, LPiston,
LPistonTarget, LPistonPeriod, LPistonDecay, LPistonTemp
}


#Set no extra parameters
set EXTRA {
#NO EXTRA PARAMETERS
}

#Set the execution script

lappend EX $mintime $T $runtime $eqtime
# Number of minimization steps, temperature and total time steps.

#Set output parameters:
```

```
set OP $outputname
lappend OP 100 100 100 100 100
#restartfreq, dcdfreq, xstfreq, outputEnergies, outputPressure


###############################################################################
```

D.14 Exec_NAMD.tcl

```
#This script checks first the number of .conf files in the folder
and execute all of these thereafter.

#Save the names of all the .conf files in the results folder to 'conflist'
set conflist [glob $FL_results*.conf]
#Sort this list on alphabeth, the previous one is random (?)
set conflists [lsort $conflist]
#Get the length of the list: this is the number of .conf files which
needs to be executed
set lconflist [llength $conflist]

#Loop over all the .conf files
for {set k 0} {$k<$lconflist} {incr k} {
        #Create a dummy variable with the current name, this also
        includes the .conf extension
        set CONFfiledum [lindex $conflists $k]
        #Create a variable with the current name of the file (throw the
        .conf extension away)
        set CONFfile [string range $CONFfiledum 0 end-5]
        #Add 1 to k, this is being used in the print status.
        set counter [expr $k +1]
        #Get the current time in yy/mm/dd-HH:MM:SS format
        set systemTime [clock seconds]
        set datetime [clock format $systemTime -format %y/%m/%d-%H:%M:%S]
        #Print the status of the simulations.
        puts "$datetime:Starting simulation $counter of $lconflist: $CONFfile"

        #Execute NAMD
        exec ${FL_namd}namd2 $CONFfile.conf > $CONFfile.log


}
```

D.15 matlabinfo.tcl

```
#This script produces a very small .txt file with all the names
 of the conffiles.

#Open a new file 'matlabinfo.txt'
set fileid [open ${FL_results}matlabinfo.txt w]

#Write the list
puts $fileid "
$conflist
"

#Close the file
close $fileid

#Open a second file 'variables.txt'
set fileid [open ${FL_results}variables.txt w]

#Write some variables to this file:
```

```
puts $fileid "
nr_sizes \t $nr_sizes
nr_rest \t $nr_rest
CONFbase \t $CONFbase
waterbase \t $waterbase
mintimeh \t $mintimeh
eqtime \t $eqtime
OPfreq  \t $OPfreq
"

close $fileid
```

D.16 hydro.tcl

```
#Calculate the number of hydrogen bonds
for {set i 0} {$i<$nr_sizes} {incr i} {
        #Delete the current loaded model
        mol delete top

        #Update the suffix tot the current file.
        set suffix _s$i

        #Open a .dat file
        set fileid [open ${FL_results}${CONFbase}${suffix}_r00.dat w]
        set fileid2 [open ${FL_results}${CONFbase}${suffix}_r00.rdf w]
        set fileid3 [open ${FL_results}${CONFbase}${suffix}_r00.den w]


        #Load the new model
        mol new ${FL_results}${waterbase}${suffix}.psf
        mol addfile ${FL_results}${waterbase}${suffix}.pdb


        for {set j 0} {$j<$nr_rest} {incr j} {
                if {$j<10} {
                                set sj 0$j
                        } else { set sj $j }
                if {$j==0} {
                        set startframe [afrond u [expr $mintimeh/$OPfreq] 0]
                        puts "$startframe"
                } else {set startframe 0}


                #Update the outputname
                set outputname $CONFbase${suffix}_r${sj}
        mol addfile ${FL_results}${outputname}.dcd first
        $startframe molid top waitfor all
        }
        #Get the number of frames
        set nr_frames [molinfo top get frame]

        for {set k 0} {$k <=$nr_frames} {incr k} {
                #Then the hbonds in DNA are needed
                set selN [atomselect top "nucleic" frame $k]
                set selW [atomselect top "water" frame $k]
                set selA [atomselect top "all" frame $k]


                #Set the cut-off parameters
                set co_l 3      ;# Cut-off length
                set co_a 20     ;# Cut-off angle
                #Get the lists of the hydrogen bonds in the system:
                #N = DNA-DNA, W=Water-Water, A=All-All, NW=DNA-Water, WN=Water-DNA
                set hbN [measure hbonds $co_l $co_a $selN]
                set hbW [measure hbonds $co_l $co_a $selW]
                set hbA [measure hbonds $co_l $co_a $selA]
                set hbNW [measure hbonds $co_l $co_a $selN $selW]
                set hbWN [measure hbonds $co_l $co_a $selW $selN]

                #The length of the previous lists equals the number of hbonds
                set nrhN [llength [lindex $hbN 0]]
                set nrhW [llength [lindex $hbW 0]]
                set nrhA [llength [lindex $hbA 0]]
                set nrhNW [llength [lindex $hbNW 0]]
                set nrhWN [llength [lindex $hbWN 0]]

                #Write the number of hydrogen bonds to a .dat file
                puts $fileid "$k \t $nrhA \t $nrhW \t $nrhN \t $nrhNW \t $nrhWN"

                #Calculate the volumes (for density calculations later on)
                #added 110330 - not same file as Urubu
                if {$sONLYWATER==1 && $sT==0} {
                        set eqtime 0
                }
                set rdfskip [afrond u [expr $eqtime/$OPfreq] 0]
                if {$k==$rdfskip || $k==[expr ceil (($nr_frames-$rdfskip)/2.0) +
                $rdfskip] || $k==$nr_frames} {
                        set selO [atomselect top "oxygen" frame $k]
                        set rdf_data [measure gofr $selO $selO delta 0.05 rmax 40]
                        if {$k==$rdfskip} {
                                puts $fileid2 [lindex $rdf_data 0]

                        }
                        puts $fileid2 [lindex $rdf_data 1]
                }

                #Get the number of particles in a small box (10^3A) around the origin
                set dimS 5
                set selS [atomselect top "(x>-$dimS and x<$dimS and y>-$dimS and
                 y<$dimS and z>-$dimS and z<$dimS) and oxygen" frame $k]
                set dimL [expr 2*$dimS]
                set selL [atomselect top "(x>-$dimL and x<$dimL and y>-$dimL and
                y<$dimL and z>-$dimL and z<$dimL) and oxygen" frame $k]
                set nr_S [$selS num]
                set nr_L [$selL num]
                puts $fileid3 "$k \t $dimS \t $dimL \t $nr_S \t $nr_L"


        }
        close $fileid
        close $fileid2
        close $fileid3

}
```

D.17 Matlab script

```
%%
%--------------------------------------------------------------------------
%This matlab macro file (.m) is able to read the output data from the NAMD
%simulations. At current state it takes the following steps:
%1) Open the matlabinfo.txt file and read the file names.
%2) Determine if any restart information is available
```

```
%3) Open the PSF files to get information about the atoms in the systems.
%4) Open the .log files, read the energies and delete the minimization
%steps.
%5) Do some calculations: the ctime is the computational time per atom per
%time step. Also get the Kinetic energy of each particle.
%6) Read the number of hydrogen bonds from a .dat file
%7) Read the radial distribution function for 3 temperatures from a .rdf
%file
%8) Choose which graps to plot. Note that, if a temperature-time fit should
%be used, the first input should be 12 (T-data), then 1(time data) and yes
%to use the fit for other graphs. In the 'more graphs' question, put yes
%and the choose the graphs required.
%--------------------------------------------------------------------------

%--------------------------------------------------------------------------
%Some basic info about Matlab and .m files:
% - a % in front of a line indicates that this line is treated as comment.
% - two %% in a single line indicate the beginning/end of a 'block' of
% code. By opening this .m file in the Matlab editor it is very easy to
% identify or execute these different blocks.
% - variables are declared with a 'x=1' similar syntax
% - Matlab is able to do several types of control flow (for, if, while,
% switch). Please type 'doc for' in the command window for more
% information.2181154
% - Where ... are used, the line is 'cut' and is continued on the next
% line. This is done for readability
% - For the purpose of saving computational time all arrays are
% preallocated as soon as their sizes are known. This is done by the
% 'x=zeros(m,n)' or the 'x=cell(m,n)' commands. Please see 'doc zeros' or
% 'doc cell' for more information.
%--------------------------------------------------------------------------


%Close all figures, clear workspase and command window.
close all
clear all
clc

%Set get the switch s.mac and s.slash based on s.mac. Any variable
%s.[something] is part of the 's' structure which is reserved for switches.
s.mac=isunix;
if s.mac==1
    s.slash='\';
else
    s.slash='/';
end

%Get output folder:
%Initialise the 's.all' variable which indicates that a outputfolder should
%be read./home/rudi/Desktop
s.all='yes';
%Repeat the whole code if 's.all' is equal to 'yes'. 's.all' is updated in
%the end of the code.
while strcmp(s.all,'yes')
    %Show the matlab desktop
    desktop

    %Load the list of most specified output folders: (if the file exist)
    if exist('outputfolders.mat','file')
        d=load('outputfolders.mat');
        d.outputnr=cell(length(d.outputfolderd),1);
        for j=1:length(d.outputfolderd)
            d.outputnr{j,1}=j;
        end
        outputfolderd=d.outputfolderd;
        d.outputprint={d.outputnr{:};outputfolderd{:}};
        d.outputprint=d.outputprint';
        format long
        display(d.outputprint);
        format short

        s.outputfolder1=0;
    else s.outputfolder1=1;
    end

    %Ask for input:
    outputfolder = input('Specify output folder: ','s');

    %Keep asking until input is provided
    while isempty(outputfolder)
        outputfolder = input('Specify output folder: ','s');
    end

    if s.outputfolder1~=1 && str2double(outputfolder)<=length(outputfolderd)
        outputfolder=outputfolderd{str2double(outputfolder),:};
    else
        if strcmp(outputfolder(end),s.slash)~=1;
            outputfolder(end+1)=s.slash;
        end


        s.outputfolderd2=input(['\n Save this directory to',...
            ' outputfolders.mat? \n',...
            '[0] = no \n[1] = yes, keep current list \n',...
            '[2] = yes, delete old list \n\n']);
        while isempty(s.outputfolderd2)
            s.outputfolderd2=input(['\n Save this directory to',...
            'outputfolders.mat? \n',...
            '[0] = no \n [1] = yes, keep current list \n',...
            '[2] = yes, delete old list \n\n']);
        end
        if s.outputfolderd2==1
            outputfolderd={outputfolderd{:};outputfolder};
           save('outputfolders.mat','outputfolderd')
        elseif s.outputfolderd2==2
            outputfolderd={outputfolder};
            save('outputfolders.mat','outputfolderd')
        end


    end

    %%
    %-------------------------------------------------------------------------
    %1) Open the matlabinfo.txt file and read the file names.
    %-------------------------------------------------------------------------

    %Open matlabinfo.txt
    fid = fopen([outputfolder 'matlabinfo.txt'],'r');

    %Display an error message (and break the simulation) if the file cannot be
    %read.
```

```
if fid==-1
    errordlg('Not able to open "matlabinfo.txt", check outputfolder')
    break
else
    desktop
    fprintf(['Succesfully opened: ',[outputfolder 'matlabinfo.txt'],'\n\n'])
end

%Create the tline variable with a random text, just to enter the loop
tline='rrr';

%While tline contains text do:
while ischar(tline)==1
    %Read a line from the file
    tline=fgetl(fid);
    %If this line is not equal to -1, then there is information
    if tline ~= -1
        %Scan the line and save it to a new variable
        outputinfo=sscanf(tline,'%s');
    end
end

%Since the folder and the extension are also included the file
%names should be filtered. Therefore get the / and . in the names
%to identify where the file names start
starts=find(outputinfo=='/');
ends=find(outputinfo=='.');
%Get the number of files by counting the entries in the variable
%'starts'
nr_files=length(starts);

%Preallocate datafiles:
datafiles = cell(nr_files,1);

%Loop over this number
for i=1:nr_files
    %Save the file name in a cell array (the names doesn't have to
    %be equal of length, therefore a cell array is used). And add
    %the .log extension, since this is the file type we'd like to
    %read.
    datafiles{i}=[outputinfo(starts(i)+1:ends(i)-1), '.log'];

end
%Sort the list on alphabeth. This list is also written to the
%command window
datafiles=sort(datafiles);
disp(datafiles)


%Close the file
fclose(fid);
%%
%--------------------------------------------------------------------------
%2) Determine if any restart information is available
%--------------------------------------------------------------------------
%Set a dummy counter, used in determining the number of restarts
c=0;

%Preallocate datatypes1:
datatypes1 = zeros(nr_files,1);
```

```
 %Loop over the datafiles and save the last index (restart index)
 for i=1:nr_files
     datatypes1(i)=str2double((datafiles{i}(end-5:end-4)));
 end
 %If the maximum index is bigger then 0, there are restarts and the number
 %of restarts is set equal to the last index (should be the biggest). If the
 %maximum index is equal to 0, there are no restarts and the number of
 %restarts is set to 0. In other cases an error message is displayed and the
 %simulation is broken.
 if max(datatypes1)>0
     nr_restarts=str2double(datafiles{end}(end-5:end-4));
 elseif max(datatypes1)==0
     nr_restarts=0;
 else
     errordlg(['Something went wrong: ' num2str(datatypes1), ' Datatypes!?'])
     break
 end

 %Loop over the datafiles again to make a vector with the index of each
 %restart. This generates a vector which identifies the systems, in a case
 %of 2 system sizes and 3 restarts this vector should be: 1 1 1 1 2 2 2 2
 %(the initial run + 3 restarts for each system).

%Preallocate restartind:
 restartind = zeros(nr_files,1);
 for i=1:nr_files
     if mod(i-1,nr_restarts+1)==0
         c=c+1;
     end
     restartind(i)=c;
 end
 %disp(restartind)
 %Set a variable which states how much structural files should be read.
 nr_open=nr_files/(nr_restarts+1);

 %%
 %-------------------------------------------------------------------------
 %3) Open the PSF files to get information about the atoms in the systems.
 %-------------------------------------------------------------------------
 %Preallocate N, Nc & atomdata:
 N=zeros(nr_open,1);
 Nc=zeros(nr_open,1);
 mass_total=zeros(nr_open,1);
 atomdata=cell(nr_open,1);
 %Loop over the number of files to open
 for i=1:nr_open
     %Display in the command window tha the next file is opened.
     disp(['open the next file ',num2str(i)])

     %Open the current psf file. Note that the 'watername' variable in TCL
     %is not saved and that the original value of this variable is just
     %'used'. This also includes the suffix, so please don't change it.
     fid = fopen([outputfolder, 'waterbox_s',num2str(i-1),'.psf'],'r');

     if fid==-1
         errordlg('Not able to open the .psf file, check outputfolder!')
         break
     else
         fprintf('Succesfully opened .psf file\n\n')
     end
```

```
    %Get the first line (doesn't contain usefull information)
    tline=fgetl(fid);

    %Keep reading the file untill the end of the file
    while ischar(tline)
        %Read a new line
        tline=fgetl(fid);
        %if this line contains more then 15 characters do some comparison
        if length(tline)>=15
            %If the 10th to 15th character make '!NATOM' start with saving
            %information (strcmp = string compare)
            if strcmp(tline(10:15),'!NATOM')
                %The number of atoms in the system is also placed in the
                %current line of the file. Just save the first 9 characters
             %and convert them to a number (it was a string).
                N(i)=str2double(tline(1:9));

                %Read atom information from the psf files in the format:
                %(1)Atom ID, (2)...,(3)charge,(4)mass, (5)...
                atomdata{i}=fscanf(fid,'%i %*s %i %*s %*s %*s %g %g %i',...
                    [5,N(i)])';

            end
        end
    end

    %Close the file
    fclose(fid);

    %Calculate the total number of internal constraitns (number of fixed
    %bonds + number of fixed bond angles)
    Nc(i)=N(i)/3*(2+1);

    %Calculate the total mass of the system, where Avo is Avogadro's
    %constant and the 1000 is the g/kg ratio.
    Avo=6.02214179e23;
    mass_total(i)=sum(atomdata{i}(:,4))/(Avo*1000);


end
%%
%--------------------------------------------------------------------------
%4) Open the .log files, read the energies and delete the minimization
%steps.
%--------------------------------------------------------------------------
%Preallocate data, timeline, startmin, realdata, combineddata
data=zeros(nr_files,1,20);
timeline=zeros(nr_files,1);
startmin=cell(nr_files,1);
realdata=cell(nr_files,1);
combineddata=cell(nr_open,1);

%Loop over all the files
for i=1:nr_files
    %Open the current datafile
    fid = fopen([outputfolder datafiles{i}],'r');
    if fid==-1
        errordlg(['Not able to open ',[outputfolder datafiles{i}],...
            ', something went wrong?'])
        break
    else
        fprintf(['Succesfully opened: ', [outputfolder datafiles{i}],...
            ',\n please be patient when this file is being read.\n\n'])
    end

    %Just put some random string in tline to get into the loop
    tline='rrr';

    %Create a dummy index which is used to save the data later on
    j=1;

    %Read the data in the format:
    %(1)TS,(2)BOND,(3)ANGLE,(4)DIHED,(5)IMPRP,(6)ELECT,(7)VDW,(8)BOUNDARY,
    %(9)MISC,(10)KINETIC,(11)TOTAL,(12)TEMP,(13)POTENTIAL,(14)TOTAL3,
    %(15)TEMPAVG,(16)PRESSURE,(17)GPRESSURE,(18)VOLUME,(19)PRESSAVG,
    %(20)GPRESSAVG
    while ischar(tline)==1
        %Get a new line
        tline = fgetl(fid);
        %Get the length of the current line, set the dummy variable einde
        %('end') to the length of this line otherwise set the einde to 7.
        %This is enough to identify if the line contains usefull
        %information.
        if length(tline)<7
            einde=length(tline);
        else
            einde=7;
        end
        %Do several things if there is a match between the tline and ENERGY
        %or WallClo. The used formats are based on:
        %  %s = a string
        %  %i = a interger
        %  %g = a floating point number
        %  a '*' indicates that this column of the line is not saved in
        %  the variable
        switch tline(1:einde)
            %If tline starts with ENERGY save this line as a data line
            case 'ENERGY:'
                dataline = sscanf(tline,['%*s %i %g %g %g %g %g %g',...
                    '%g %g %g %g %g %g %g %g %g %g %g %g'],[21 1])';
                data(i,j,:)=dataline;
                j=j+1;
                %If tline starts with WallClo save this line as a time line
            case 'WallClo'
                timeline(i) = sscanf(tline, '%*s %g %*s %*g %*s %*g %*s');

        end
    end
    %Close the opened file
    fclose(fid);

    %Delete the minimization steps:
    %Find the non-zero entries in the temperature data
    startmin{i}=find(data(i,:,12)~=0);
    %Save this data to a new variable 'realdata'
    realdata{i}(:,:)=data(i,startmin{i}(:),:);


    if (mod(i-1,nr_restarts+1)==0)
        %If there is no combined data, make a new one.
        realdata{i}(:,1)=realdata{i}(:,1)-realdata{i}(1,1);
        combineddata{restartind(i)}(:,:)=realdata{i}(:,:);
    elseif (mod(i-1,nr_restarts+1)==1)
        %If there is, and this is the first restart, add all but the last
        %ts information to the combineddata.
        combineddata{restartind(i)}=...
            [(combineddata{restartind(i)}(:,:));realdata{i}(1:(end-1),:)];
    else
        %If there is, and this is not the first restart, add the real data
        %of the restart simulations to the combineddata.
        combineddata{restartind(i)}=[(combineddata{restartind(i)}(:,:));...
            realdata{i}(:,:)];
    end



end

for i=1:nr_open
    %Delete duplicates made by heating protocol
    duplicates=find(combineddata{i}(1:end-1,1)==combineddata{i}(2:end,1));
    for j=1:length(duplicates)
        combineddata{i}=[combineddata{i}(1:duplicates(j)-1,:);...
            combineddata{i}(duplicates(j)+1:end,:)];
    end

    %Rescale the timesteps to a dimensionless time
    combineddata{i}(:,1)=combineddata{i}(:,1)/combineddata{i}(end,1);
end

%Get the size of the combined data= #ts, nr of colums in combineddata
ammountcdata=size(combineddata{1});
%%
%-------------------------------------------------------------------------
%5) Do some calculations: the ctime is the computational time per atom per
%time step. Also get the Kinetic energy of each particle.
%-------------------------------------------------------------------------
%Preallocate ctime, T
ctime = zeros(nr_open,1);
T=zeros(nr_open,ammountcdata(1));
Ekin=zeros(nr_open,ammountcdata(1));
temps=zeros(nr_open,1);
for i=1:nr_open
    ctime(i)=sum(timeline(1+(nr_restarts+1)*(i-1):i*(nr_restarts+1)))/...
        (N(i)*combineddata{i}(end,1));

    %Extract the kinetic energies:
    T(i,:)=combineddata{i}(:,12);
    Ekin(i,:)=combineddata{i}(:,10);
    nr_zero=find(combineddata{i}(:,12)==0);
    lcombineddata=size(combineddata{i});
    temps(i)=sum(combineddata{i}(:,12))/(lcombineddata(1));
end
fprintf(['Computational time per atom per ts = ',num2str(ctime)])

%%
%-------------------------------------------------------------------------
%6) Calculate the number of hydrogen bonds in the system.
%-------------------------------------------------------------------------
%Preallocate nr_at, hbonds
nr_at=zeros(nr_open,1);
hbonds=cell(nr_open,1);
```

```
hbondssize=zeros(nr_open,2);

for i=1:nr_open
    %Get the number of atom types in the current system (to be able to
    %check for water/DNA)
    nr_at=length(unique(atomdata{i}(:,4)));

    %Change the title of the plots based on the system.
    if nr_at>2
        titletext='DNA';
    elseif nr_at==2
        titletext='Waterbox';
    else
        %Break the .m file if nr_at is not >=2
        errordlg(['Something went wrong: ', num2str(nr_at(i)),...
            ' atomtypes?'])
        break
    end

    %Open the .dat files which contain the hbonds data.
    fid = fopen([outputfolder,datafiles{i*(1+nr_restarts)}(1:end-4),...
        '.dat'],'r');
    %Break the code if the .dat files are not opened (change 'break' into '
    %continue' if you'd like to do a simulation without hbonds
    %calculations.
    if fid==-1
        errordlg(['Matlab was not able to open .dat',...
            ' check outputfolder!']);
        break
        %continue
    else
        %Read the hbonds information, save them in a cell array.%
        hbonds{i}=load ([outputfolder,...
            datafiles{i*(1+nr_restarts)}(1:end-4),'.dat']);
        hbondssize(i,:)=size(hbonds{i});
        if hbondssize(i,2)==2
            hbonds{i}(end,6)=0;
        end
    end

    %It is possible that VMD supplies not enough hbonds data, reason
    %unknown. Therefore, the last data enry in hbonds will be copied if
    %that's the case.
   if length(hbonds{i})==(length(combineddata{i})-1)
        hbonds{i}(end+1,:)=hbonds{i}(end,:);
        fprintf(['\n Length hbonds ~= to length combineddata, last',...
            'last hbonds data is copied! \n\n'])
    end

end

%%2478
%-------------------------------------------------------------------------
%7) Import and save the radial pair distribution function data.
%-------------------------------------------------------------------------
rdf_data=cell(nr_open,1);
for i=1:nr_open
    if exist([outputfolder,...
            datafiles{i*(1+nr_restarts)}(1:end-4),'.rdf'],'file')
        rdf_data{i}=importdata([outputfolder,...
            datafiles{i*(1+nr_restarts)}(1:end-4),'.rdf']);
        rdf_data{i}=rdf_data{i}';
    else
        errordlg('The .rdf file is not found. Please check outputfolder!')
        continue
    end
end
%%
%---------------------------------------------------------------------------
%8) This part of the code allows the user to actively request some plots.
%Note that the command window shows up after running this part (or the
%whole code) where input is requested!
%---------------------------------------------------------------------------

%Preallocate legendtext
legendtext=cell(nr_open,1);

%Make a colormap based on the number of system sizes. The additional one
%is used to indicate the restart lines.
figure
cmap=colormap(jet(nr_open+1));
cmap2=colormap(jet((nr_open)*5));
%Since 'colormap' opens a figure this 1 should be closed again.
close(gcf)

%Preallocate a switch for plot infos
plotinfos=0;

while plotinfos==0

    %Show the command window
    desktop
    %Ask the user to give plot information
    %Y axis
    plotoption=['(1)TS,(2)BOND,(3)ANGLE,(4)DIHED,(5)IMPRP,',...
        '(6)ELECT,(7)VDW,(8)BOUNDARY,\n',...
        '(9)MISC,(10)KINETIC,(11)TOTAL,(12)TEMP,(13)POTENTIAL,(14)TOTAL3,\n'...
        '(15)TEMPAVG,(16)PRESSURE,(17)GPRESSURE,(18)VOLUME,(19)PRESSAVG\n,',...
        '(20)GPRESSAVG, !!!(21) DENSITY, (22) TEMP HIST, (23) HBONDS,',...
      '(24) HBONDS (ZOOMED),(25) RDF\n\n'];

    plotinfoy = input(['\n Which value should be plotted on the y axis? \n',...
        plotoption,...
        '(Please provide 1 number or vector: [a,b,c])\n\n']);
    %Keep requesting for input untill it is provided.
    while isempty(plotinfoy)
        plotinfoy = input(['\n Which value should be plotted on the y axis? \n',...
            plotoption,...
            '(Please provide 1 number or vector: [a,b,c])\n\n']);
    end

    %X axis: (don't afk if plotinfoy==25, RDF data is used different)
    if length(plotinfoy)>1 || plotinfoy~=25
        plotinfox = input(['\n Which value should be plotted on the x axis? \n',...
            plotoption,...
            '(Please provide 1 number (all plots get the same x axis)',...
            ' or a vector of the length of y)\n\n']);
        %Keep requesting for input untill it is provided.
        while isempty(plotinfox)
        plotinfox = input(['\n Which value should be plotted on the x axis? \n',...
                plotoption,...
                '(Please provide 1 number (all plots get the same x axis)',...
                ' or a vector of the length of y)\n\n']);
        end
        if (length(plotinfoy)~=length(plotinfox) && length(plotinfox)~=1)
     errordlg(['Number of y entries is not equal to the number of x entries',...
                '. Please provide an equal number of entries or provide 1 x',...
                ' entry, this will plot all the plots with the same x axis'])
        end
    else
        plotinfox = 0;
    end

    %This is a cell array with strings, used to make the ylabels (and titles)
    %at each graph.
    ylabeltext={'Time [-]','Bond [?]','Angle [?]','Dihedrals [?]',...
        'Impropers [?]','Elect? [?]','Van der Waals [?]','Boundary? [kcal/mol]',...
       'Misc? [kcal/mol]','Kinetic Energy [kcal/mol]','Total Energy [kcal/mol]',...
        'Temperature [K]','Potential Energy [kcal/mol]','Total3? [kcal/mol]',...
        'Average Temperature [K]','Pressure [bar]','GPressure [bar]',...
        'Volume [A^3]','Average Pressure [bar]','Average GPressure [bar]',...
        'NP Density [kg/m^3]','Relative frequency [-]',...
        '# of Hydrogen bonds [-]','# of Hydrogen bonds (ZOOMED) [-] ',...
        'g(r) [-]'};
    xlabeltext=ylabeltext;
    xlabeltext{25}='r/\sigma [-]';


    %Loop over the length of input.
    for p=1:length(plotinfoy)
        clear legendtext
        legendtext=cell(1);
        legendtext{1}='Initial legendtext';
        %Determine x axis index
        if length(plotinfox)==1
       xindex=plotinfox;
        else
            xindex=plotinfox(p);
        end

        %Determine if the T-t fit should be used for the next plots
        if exist('sfit','var')
            if strcmp(sfit,'yes') && xindex == 12
                %Generate a vector with temperatures based on the ts
                %data.
                xdata=lfit(combineddata{i}(:,1));
            end
        else
            xdata=combineddata{i}(:,xindex);
        end


        %Determine plot type on xindex (all non TS choices should be
        %represented by crosses (no lines))
        if xindex~=1
            plottype='x';
        else
            plottype='-';
        end
        %Do different things for different plots
        switch plotinfoy(p)
            %The first 20 plots are provided by the same output which is used
            %in the same way for all plots.
```

```
case {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20}
    figure
    for i=1:nr_open
        hold on
        plot(xdata,...
            combineddata{end-i+1}(:,plotinfoy(p)),...
            plottype,'Color',cmap(i,:))
        legendtext{i,:}=['N=',num2str(N(end-i+1))];


    end

    ylim=get(gca,'Ylim');
    for j=1:nr_restarts
        restartts=realdata{j}(end,1);
        line([restartts,restartts],ylim,'Color',cmap(end,:))
    end

    if nr_restarts>1
        legendtext{i+1,:}='Restart points';
    end

    switch plotinfoy(p)
        case 12
            warning off
            f=fittype('a*x+b','independent','x');
            lfit=fit(combineddata{i}(:,xindex),...
                combineddata{i}(:,plotinfoy(p)),f);
            plot(lfit,'k')
            legendtext{end+1,:}='Linear fit of T';
         legend('off')
            warning on
            desktop
            sfit=input(['\n Use this fit for the other',...
                'plots? [yes/no] \n\n'],'s');
            while isempty(sfit)
                sfit=input(['\n Use this fit for the other',...
                    'plots? [yes/no] \n\n'],'s');
            end



    end
    ylabel(ylabeltext{plotinfoy(p)})
    xlabel(xlabeltext{xindex})
    title([ylabeltext{plotinfoy(p)},' of ', num2str(nr_open),...
        ' ',titletext,' simulations, with ',...
        num2str(nr_restarts),' restarts per sim.'])
    legend(legendtext,'Location','SE')
    titlestopy = find(ylabeltext{plotinfoy(p)}=='[');
    titlestopx = find(xlabeltext{xindex}=='[');
    saveastext=[outputfolder,...
        ylabeltext{plotinfoy(p)}(1:titlestopy-1),...
        ' versus ',xlabeltext{xindex}(1:titlestopx-2)];

case 22
    %Plot temperature histogram:
    for i=1:nr_open
        figure
        hold on
        [n,xout]=hist(combineddata{i}(:,12),20);
        n=n/(max(n));
        f=fittype('gauss1');
        gfit = fit(xout',n',f);
        bar(xout,n)
        plot(gfit)
        ylabel(ylabeltext{plotinfoy(p)})
        xlabel('Temperature [K]')
        title('Histogram of temperature distributions')
        legend(['N=',num2str(N(i))],'Gaussian fit','Location','NW')
        saveastext = [outputfolder,...
            'Temperature distribution'];
    end
case {23, 24}
    %Plot number of bonds
    for i=1:nr_open
        figure
        hold on
        if (isempty(hbonds{i})==0 && xindex<=20)
            if plotinfoy(p)==23
                lcounter=5;
                lshift=0;
                plot(combineddata{i}(1:(end),xindex),hbonds{i}(:,2),...
                    plottype,'Color',cmap2((i-1)*lcounter+1-lshift,:))
      plot(combineddata{i}(1:(end),xindex),hbonds{i}(:,3),...
                    plottype,'Color',cmap2((i-1)*lcounter+2-lshift,:))
                legendtext{(i-1)*lcounter+1-lshift,:}=['N= ',
                 num2str(N),...
                    ', All bonds'];
                legendtext{(i-1)*lcounter+2-lshift,:}=['N= ',
                num2str(N),...
                    ', WW bonds'];
                saveastext = [outputfolder,...
                    'Number of hydrogen bonds in the system'];
            else
                lcounter=3;
                lshift=2;
                saveastext = [outputfolder,...
                    'Number of hydrogen bonds in the system,
                    (ZOOMED)'];
            end
            plot(combineddata{i}(1:(end),xindex),hbonds{i}(:,4),...
                plottype,'Color',cmap2((i-1)*lcounter+3-lshift,:))
            plot(combineddata{i}(1:(end),xindex),hbonds{i}(:,5),...
                plottype,'Color',cmap2((i-1)*lcounter+4-lshift,:))
            plot(combineddata{i}(1:(end),xindex),hbonds{i}(:,6),...
                plottype,'Color',cmap2((i-1)*lcounter+5-lshift,:))

            legendtext{(i-1)*lcounter+3-lshift,:}=['N= ',
            num2str(N),...
                ', DD bonds'];
            legendtext{(i-1)*lcounter+4-lshift,:}=['N= ',
            num2str(N),...
                ', WD bonds'];
            legendtext{(i-1)*lcounter+5-lshift,:}=['N= ',
            num2str(N),...
                ', DW bonds'];


        else
            errordlg(['Not possible to plot 22 against 21
            or 22 or hbondssize:', num2str(hbondssize(1)), ' is 0'])
            legendtext=0;
```

```
                        continue

                    end
                end
                xlabel(xlabeltext{xindex})
                ylabel(ylabeltext{plotinfoy(p)})
                title(['Number of hydrogen bonds per ',xlabeltext{xindex}])
                legend(legendtext)
            case 25
                for i=1:nr_open
                    figure
                    hold on
                    plot(rdf_data{i}(:,1),rdf_data{i}(:,2))
                    plot(rdf_data{i}(:,1),rdf_data{i}(:,3),'r')
                    plot(rdf_data{i}(:,1),rdf_data{i}(:,4),'g')
                    ylabel('g(r)')
                    xlabel('r/\sigma')
                    title('Radial distribution function')
                    T1=round(combineddata{i}(1,12)/10)*10;
                    T2=round(combineddata{i}(ceil(ammountcdata(1)/2),12)/10)*10;
                    T3=round(combineddata{i}(end,12)/10)*10;
                   legend(['T \approx ', num2str(T1),'K'],...
                        ['T \approx ', num2str(T2),'K'],...
                        ['T \approx ', num2str(T3),'K'])
                    xindex=plotinfoy(p);
                    saveastext = [outputfolder,...
                        'Radial distribution function'];
                end
            otherwise
                errordlg(['Plot info was not understood try 1 number or',...
                    'vector: [a,b,c]'])
        end
        %Save the figures to .png's
        saveas(gcf,saveastext,'png')
    end
    %Ask for more plots
    desktop
    plotinfos2=input('\n Generate more plots? [yes/no] \n\n', 's');
    if strcmp(plotinfos2,'yes')
        plotinfos=0;
    else
        plotinfos=1;
    end
end
s.all = input('\n Want to open aditional files? [yes/no] \n\n','s');
while isempty(s.all)
    s.all = input('\n Want to open aditional files? [yes/no] \n\n','s');
end
end
```