

"A Mesoscopic Computer Model for Reinforcement in Filled and Strain-Crystallizing Elastomers" - Manual for the Computer Program

Lena Tarrach
Bergische Universität Wuppertal

The computer program is employed for the simulation of the uniaxial deformation of filled and strain-crystallizing elastomer networks. It is written in C++. The program code consists of the following files:

- `main.cpp`,
- `makefile`,
- `network.hpp`,
- `helper.hpp`,
- `randomnumbers.h`,
- `ucomm.hpp`.

Parameters depending on the desired network configuration are defined in `main.cpp`, whereas fixed parameters are given in `network.hpp`. The latter file includes the necessary functions for the simulations. The `makefile` is used for compilation of the code. In the remaining header files, functions required for certain computations are included. The folder `header` contains all of the header files.

Computations of the tensile strength are conducted in the Jupyter Notebook `TensileTestAnalysis.ipynb`. In the Jupyter Notebook `FatigueAnalysis.ipynb` the data for the Wöhler curves are obtained.

Subsequently, the essential functionalities of the program are described. Note that there are unused variables or functions in the current version of the program because it has been progressively developed for different investigations of the model elastomer networks.

Contents

1	The Main File <code>main.cpp</code>	1
2	The Header File <code>network.hpp</code>	3
3	Additional Header Files	13

1 The Main File `main.cpp`

In the main file, the values for the variables given in Table 1 are set and the functions given in Table 2 are called to run the simulation of elastomer networks. The essential parameters characterizing the model elastomer network are specified in this file.

When reading the initial configuration from a file, note that `initialFiller.txt` is not required because it is not used.

Table 1: Table of fixed quantities used in the code in `main.cpp` in alphabetic order.

Variable	Type	Description
<code>crystallinity_theshold</code>	<code>const double</code>	If the crystallinity of a link exceeds this value, the link cannot break. <i>This is not used in the current version of the program.</i>
<code>cutoff</code>	<code>const double</code>	cut-off radius for
<code>cycles</code>	<code>const size_t</code>	number of strain cycles
<code>defects</code>	<code>const double</code>	fraction of links in a crystallizable network which is non-crystallizable
<code>dimension</code>	<code>const size_t</code>	the dimension of the simulated network
<code>energy_histogram</code>	<code>const bool</code>	energy histograms are obtained at certain stretches if this is <code>true</code>
<code>eta</code>	<code>const double</code>	interaction strength of semi-crystalline links
<code>filled</code>	<code>const bool</code>	the network is filled if this is <code>true</code>
<code>filler_bonds</code>	<code>const size_t</code>	number of nodes a filler nodes is connected with, <i>this is not used in the current version of the program</i>
<code>hole</code>	<code>const bool</code>	inserts a hole in the center of the network if this is <code>true</code>
<code>hole_size</code>	<code>const double</code>	Size of an artificially inserted hole in units of approximately twice the initial distance of the nodes. It depends on the number of links deleted for setting up the hole. <i>This is not used in the current version of the program.</i>
<code>kappa</code>	<code>const double</code>	parameter for perturbation of initial lattice
<code>local_link_data</code>	<code>const bool</code>	
<code>max_stretch</code>	<code>const double</code>	maximum stretch in a strain cycle
<code>read_initial_network</code>	<code>const bool</code>	the initial positions of the nodes, their type, their equilibrium distances and the number of Kuhn segments per link are read from different files if this is <code>true</code>
<code>runsN</code>	<code>const size_t</code>	number of parallel runs of the simulation (for each of these runs a network is set up)
<code>rupture</code>	<code>const bool</code>	the critical free energy density is applied to the links if this is <code>true</code>
<code>save_positions</code>	<code>const bool</code>	positions of the nodes are saved if this is <code>true</code>
<code>stretch_increment</code>	<code>const double</code>	increment with which the stress is incremented or decremented
<code>stretching_only</code>	<code>const bool</code>	If this is <code>true</code> , the network is stretched. If this is <code>false</code> , retraction begins at the maximum stretch until the initial stretch is reached, i.e. a strain cycle is performed.

Table 2: Table of functions used in the code in `main.cpp` in alphabetic order.

Function	Type	Description
<code>main()</code>	<code>void</code>	In this function the critical free energy density <code>rupture_energy</code> and the fraction <code>filler_content</code> of nodes which are filler are set. It is also set whether the network is crystallizing or non-crystallizing. These parameters can be given to the function as <code>userinput</code> . Each run is executed in a different thread by calling <code>simulate_network</code> .
<code>simulate_network(_run_idx, _rupture_energy, _crystallizability, _filler_content)</code>	<code>void</code>	This function runs the simulation of a single network by using functions from the <code>network.hpp</code> -file. The number of MC steps per node <code>MC_attempts</code> of the MG is set here based on the dimensionality of the network. It is also set whether the number of Kuhn segments in the links is saved in a file.

2 The Header File `network.hpp`

The header file `network.hpp` contains the class `network`. For the computations in course of the simulation of an elastomer network, the variables given in Table 3 and the functions given in Table 4 are defined. The function `run_simulation()` is the core of the simulation algorithm.

For the simulation of identically configured networks, data can be saved to txt-files. Their format corresponds to that required when reading the initial configuration.

Table 3: Table of variables used in the code in `network.hpp` in alphabetic order. The parameters of the FIRE algorithm are omitted here.

Variable	Type	Description
<code>A0_inv</code>	<code>double</code>	Inverse of the original cross section of the simulation box.
<code>box_size</code>	<code>vector<double></code>	Vector containing the lengths of the simulation box in each spatial direction.
<code>chi</code>	<code>double</code>	Crystallinity index.
<code>Connections</code>	<code>vector<vector<size_t>></code>	Vector containing the indices of the two nodes which are connected and the type of the linkage, i.e. 0=polymer-polymer, 1=filler-filler, 2=polymer-filler
<code>CrackArea</code>	<code>vector<bool></code>	Vector containing the information whether a node is located in the region around the artificially inserted hole. <i>This is not used in the current version of the code.</i>

Continued on next page

Table 3 - continued from previous page

Variable	Type	Description
CrackCheck	bool	A hole is artificially inserted into the initial network if this is true . <i>This is not used in the current version of the code.</i>
CrackCentered	bool	An artificially inserted hole is located in the center of the model network if this is true . Otherwise, the crack is located at the boundary of the network. <i>This is not used in the current version of the program.</i>
CrackSize	double	Size of an artificially inserted hole in units of approximately twice the initial distance of the nodes. It depends on the number of links deleted for setting up the hole. <i>This is not used in the current version of the program.</i>
crystallizing	bool	A network is crystallizable if this is true .
Crystals	vector<double>	Vector containing the number c of crystalline segments in a link.
cut_ff	bool	Filler-Filler bonds break beyond a certain cut-off distance if this is true .
cut_pf	bool	Polymer-Filler bonds break or weaken (depending on spring_pf_weak) beyond a certain cut-off distance if this is true .
Defects	vector<vector<bool>>	Vector containing the information whether a link is crystallizable. <i>This is not used in the current version of the code.</i>
defect_frac	double	Fraction of non-crystallizable links. <i>This is not used in the current version of the code.</i>
d_init	const double	Initial distance between nearest-neighbor nodes.
eq_dist	vector<vector<double>>	Vector containing the equilibrium distances of the links and bonds, the cut-off radii of the bonds and whether the nodes are currently connected or not (due to reversible bond breaking for filler-filler and polymer-filler bonds).

Continued on next page

Table 3 - continued from previous page

Variable	Type	Description
<code>eta</code>	<code>double</code>	Parameter for the strength of the interaction between semi-crystalline links.
<code>eta_c</code>	<code>double</code>	<code>eta · lc</code>
<code>filled</code>	<code>bool</code>	The network is filled if this is <code>true</code> .
<code>Filler</code>	<code>vector<bool></code>	Vector containing information whether a node is filler or a cross link. A node is filler if the corresponding entry is <code>true</code> .
<code>filler_bonds</code>	<code>size_t</code>	Number of bonds of a filler node. <i>This is not used in the current version of the code.</i>
<code>filler_frac</code>	<code>double</code>	Fraction of nodes considered as filler.
<code>filler_neighbors</code>	<code>vector<vector<size_t>></code>	Vector containing the neighbor list of each node which is used in the MG.
<code>filler_rcut</code>	<code>const double</code>	Cut-off radius for establishing bonds involving filler nodes.
<code>filler_rcut_sq</code>	<code>const double</code>	<code>filler_rcut · filler_rcut</code>
<code>Force</code>	<code>vector<vector<double>></code>	Vector containing the coordinates of the force vectors acting on each node.
<code>Force_abs</code>	<code>vector<vector<double>></code>	Vector containing the absolute values of the coordinates of the force vectors acting on each node.
<code>force_hist_lambda</code>	<code>vector<double></code>	Stretches at which the free energy densities of the links are saved for obtaining a histogram.
<code>Free_Energy</code>	<code>vector<double></code>	Vector free energy of each link and bond.
<code>G</code>	<code>double</code>	Total free energy of the network.
<code>init_dist</code>	<code>const size_t</code>	Initial distance between nearest-neighbor nodes.
<code>init_length</code>	<code>double</code>	Initial length of the square or cubic simulation box.
<code>interface_area</code>	<code>const double</code>	Effective contact area a at an interface.
<code>interface_polymer_filler_factor</code>	<code>const double</code>	Prefactor for evaluation of the Metropolis criterion in the MG: <code>interface_tension_polymer_filler · interface_area</code> .
<code>interface_tension_polymer_filler</code>	<code>const double</code>	Interface tension of polymer and filler according to OWRK-theory.

Continued on next page

Table 3 - continued from previous page

Variable	Type	Description
<code>kappa</code>	<code>double</code>	A measure for the perturbation of the initially homogeneous lattice.
<code>lc</code>	<code>const double</code>	Number of Kuhn segments per crystalline segment.
<code>LinkCheck</code>	<code>vector<bool></code>	If this is <code>true</code> for a particular link, the end-to-end distance of this link is larger than its contour length, i.e. $r > n$. The vector contains the information for each link.
<code>Link_Centers</code>	<code>vector<vector<double>></code>	Vector containing the positions of the midpoints of each link.
<code>Link_Neighbors</code>	<code>vector<vector<size_t>></code>	Vector containing the indices of the neighboring links for each link.
<code>LinksN</code>	<code>size_t</code>	Total number of links.
<code>LinksCracked</code>	<code>size_t</code>	Absolute number of broken links.
<code>LinksCrackedRN</code>	<code>size_t</code>	Number of broken links with $r > n$.
<code>Links_Node</code>	<code>size_t</code>	Number of Links per node estimated based on the dimension of the network.
<code>Links_per_Node</code>	<code>double</code>	Number of Links per node computed based on the actual initial lattice.
<code>local_link_data</code>	<code>vector<vector<double>></code>	Vector containing stretch, crystallinity, extension, and free energy of each link (not polymer-filler and filler-filler bonds). <i>This is not used in the current version of the code.</i>
<code>local_link_data_save</code>	<code>bool</code>	Stretch, crystallinity, extension, and free energy of each link (not polymer-filler and filler-filler bonds) are saved if this is <code>true</code> .
<code>make_force_hist</code>	<code>bool</code>	The free energy density of each link is saved at certain stretches for obtaining a free energy density histogram if this is <code>true</code> .
<code>max_strain</code>	<code>double</code>	Maximum stretch.
<code>Neighbors</code>	<code>vector<vector<size_t>></code>	Vector containing the indices of the neighboring nodes of each node.

Continued on next page

Table 3 - continued from previous page

Variable	Type	Description
MCattempts	size_t	Number of MC interchange attempts per node for the computation of the total number of MC steps of the MG.
MCsteps	size_t	Total number of MC steps of the MG.
Nodes	vector<vector<double>>	Vector containing the coordinates of the positions of the nodes.
NodesN	size_t	Initial number of nodes.
nodes_rcut	const double	Cut-off radius used for linking nodes.
nodes_rcutsq	const double	$\text{nodes_rcut} \cdot \text{nodes_rcut}$
no_ff	size_t	Number of filler-filler bonds.
no_pf	size_t	Number of polymer-filler bonds.
Periodic_BC	vector<vector<vector<double>>>	Vector containing the information whether a link crosses the boundary of the simulation box.
r0	const double	Characteristic distance for the interaction of semi-crystalline links.
r0_inv	const double	$1/\text{r0}$
ratio_A1A2	double	Ratio of the cross sections A_1 and A_2 of the sample depending on the stretch.
rcut	double	Cut-off radius for the interaction of semi-crystalline links.
rcutsq	double	$\text{rcut} \cdot \text{rcut}$
read_initial_config	bool	The initial network is set up based on a configuration that is given by certain files which can be read.
rm	double	"Skin" radius for the neighbor list of links.
rmsq	double	$\text{rm} \cdot \text{rm}$
RuptureCheck	bool	Links can break if the rupture criterion is fulfilled and if this is true .
RuptureCryst	double	Maximum crystallinity that is allowed for links which can break. <i>This is not used in the current version of the program.</i>
RuptureEnergy	double	Critical free energy density for rupture of links.
R_factor_ff	const double	Factor R^{ff} for the computation of the cut-off radius of filler-filler bonds.

Continued on next page

Table 3 - continued from previous page

Variable	Type	Description
<code>R_factor_ff_sq</code>	<code>const double</code>	$R_factor_ff \cdot R_factor_ff$
<code>R_factor_pf</code>	<code>const double</code>	Factor R^{pf} for the computation of the cut-off radius of polymer-filler bonds.
<code>R_factor_pf_sq</code>	<code>const double</code>	$R_factor_pf \cdot R_factor_pf$
<code>r_morphology</code>	<code>const double</code>	Radius r_{MG} used for setting up the neighbor lists of the nodes used in the MG.
<code>r_morphology_sq</code>	<code>const double</code>	$r_morphology \cdot r_morphology$
<code>save</code>	<code>bool</code>	Data required for plotting the network is saved if this is true.
<code>Save_Segments</code>	<code>bool</code>	The number of Kuhn segments in each link is saved if this is true .
<code>Segments</code>	<code>vector<vector<double>></code>	Vector containing the number of Kuhn segments for each link.
<code>sigma</code>	<code>double</code>	Engineering stress.
<code>sigma1</code>	<code>double</code>	First contribution to the engineering stress.
<code>sigma2</code>	<code>double</code>	Second contribution to the engineering stress.
<code>spring_ff</code>	<code>const double</code>	Spring constant k^{ff} for filler-filler bonds.
<code>spring_ff_half</code>	<code>const double</code>	$0.5 \cdot spring_ff$
<code>spring_ff_inv</code>	<code>const double</code>	$3.0/spring_ff$
<code>spring_pf</code>	<code>const double</code>	Spring constant k^{pf} for polymer-filler bonds.
<code>spring_pf_half</code>	<code>const double</code>	$0.5 \cdot spring_pf$
<code>spring_pf_inv</code>	<code>const double</code>	$3.0/spring_pf$
<code>spring_pf_weak</code>	<code>const double</code>	Spring constant k_{weak}^{pf} for polymer-filler bonds.
<code>spring_ff_weak_half</code>	<code>const double</code>	$0.5 \cdot spring_pf_weak$
<code>strain_steps</code>	<code>double</code>	Steps in which the stretch is changed during the deformation.
<code>surface_tension_filler</code>	<code>const double</code>	Surface tension of the filler.
<code>surface_tension_filler_d</code>	<code>const double</code>	Disperse contribution to the surface tension of the filler.
<code>surface_tension_filler_p</code>	<code>const double</code>	Polar contribution to the surface tension of the polymer.
<code>surface_tension_polymer_d</code>	<code>const double</code>	Surface tension of the polymer.
<code>surface_tension_polymer_d</code>	<code>const double</code>	Disperse contribution to the surface tension of the filler.
<code>surface_tension_polymer_p</code>	<code>const double</code>	Polar contribution to the surface tension of the polymer.
<code>theta0</code>	<code>const double</code>	Free energy required for SIC.
<code>theta0_c</code>	<code>const double</code>	$theta0 \cdot lc$
<code>use_morphology_generator</code>	<code>bool</code>	The morphology generator is applied if this is true .

Table 4: Table of functions used in the code in `network.hpp` in alphabetic order.

Function	Type	Description
<code>crystallinity_index()</code>	<code>void</code>	The crystallinity of the network is computed.
<code>crystallization_check()</code>	<code>void</code>	This function includes the local free energy minimization in which it is checked whether the free energy of a link is minimized by SIC.
<code>filler_reversible_bonds()</code>	<code>void</code>	It is checked for the filler-filler bonds and polymer-filler bonds whether their end-to-end distance exceeds their cut-off distance. This function is called after energy minimization by the FIRE algorithm.
<code>FIRE_optimizer.morphology()</code>	<code>void</code>	The free energy of the network is minimized by using the FIRE algorithm. The forces and energies are computed here by calling <code>force_and_energy()</code> .
<code>force_and_energy(allow_rupture, save_forces, force_hist_index)</code>	<code>void</code>	Forces and energies of links and bonds in the network are computed. Based on that, the rupture criterion is examined by calling <code>link_rupture_morphology()</code> if rupture of links is allowed. Data for free energy histograms can be saved here.
<code>initial_FIRE_optimizer()</code>	<code>void</code>	The free energy of the initial network, which only consists of links, is minimized by using the FIRE algorithm. The forces and energies are computed here by calling <code>initial_force_and_energy()</code> .
<code>initial_force_and_energy()</code>	<code>void</code>	Force and energy in the network are computed based on the initial configuration, in which all linkages represent polymer. SIC and rupture of links are omitted.
<code>initial_lattice(_read_init_config, _node_positions_file, _linkage_file, _filler_file)</code>	<code>void</code>	First, the number of nodes is adjusted in order to fit a square or cubic lattice. The size of the simulation box and the total number of MC steps of the MG are computed based on that. The nodes are placed on the lattice which is then randomized. Afterwards the MG is applied for filled networks. Neighbor lists of the nodes are obtained and links are established. If the initial configuration is read from files instead, the positions of the nodes and linkages are obtained from these.

Continued on next page

Table 4 - continued from previous page

Function	Type	Description
<code>insert_crack()</code>	void	An artificial hole is inserted into the network by deleting a certain number of linkages at the boundary. <i>This function is not used in the current version of the program.</i>
<code>insert_crack_centered()</code>	void	An artificial hole is inserted into the network by deleting a certain number of linkages in the center. <i>This function is not used in the current version of the program.</i>
<code>link_center_positions_morphology()</code>	void	The midpoints of the links are computed.
<code>link_rupture_morphology(_g_n, node_index, neighbor_index, link_index, _link_crystallinity)</code>	bool	The rupture criterion is evaluated. If a link breaks, it is deleted from the network.
<code>links_fixed_neighbor_list_morphology()</code>	void	The neighbor list for each link is obtained which is applied in the computation of the interaction of semi-crystalline links. For this purpose the configuration at the maximum stretch is considered.
<code>local_free_energy(c, n, link_index, j, k, neigh_index)</code>	double	The free energy of a link is computed. This function is called in <code>crystallization_check()</code> when energy minimization due to SIC is examined.
<code>morphology_energy()</code>	double	The total surface free energy of the network is computed which is used to monitor the MG.
<code>nodes_neighbor_list()</code>	void	The links and bonds are established between the nodes. If an artificial hole must be inserted, this is done in this function.
<code>read_initial_filler(_initial_filler_file)</code>	void	This function reads the type of the nodes from a file, i.e. whether they are filler or not.
<code>read_initial_links(_initial_linkage_file)</code>	void	This function reads the initial links from a file and assigns the type "filler" to certain nodes according to <code>read_initial_filler()</code> . The equilibrium distances of filler-filler and polymer-filler bonds are also determined. They are given by the current distances of the nodes. <i>This function is not used in the current version of the code.</i>

Continued on next page

Table 4 - continued from previous page

Function	Type	Description
<code>read_initial_node_positions(_initial_positions_file)</code>	void	The positions of the nodes are read from a file. The positions have to correspond to those after the first energy minimization. The linkages are set up in <code>read_initial_links()</code> , where also filler nodes are defined.
<code>run_morphology_generator()</code>	void	This function is the MG. First, the functions <code>set_filler_nodes()</code> and <code>set_morphology_neighbors()</code> are called. Then, the MC algorithm follows.
<code>run_simulation(_max_strain, _strain_steps, _run, _cycles, _stretching_only)</code>	void	The energy is minimized and the network is deformed until the maximum stretch is reached. If enabled, the network is then retracted until the initial stretch is reached again. In particular, the stress and the crystallinity at each stretch are saved in a file.
<code>save_all_local_data(_idx)</code>	void	Stretch, extension and free energy density of each link can be saved. This function is called in <code>save_data()</code> .
<code>save_data()</code>	void	Positions of the nodes, size of the simulation box, and information about the crystallinity of each link can be saved.
<code>save_initial_r_of_links()</code>	void	The initial end-to-end distance of each link is saved.
<code>save_Kuhn_segments(_save_Segments)</code>	void	It is defined whether the number of Kuhn segments in each link is saved in a file.
<code>set_approx_nodes(_nodes)</code>	void	The total number of nodes in the network is defined.
<code>set_crack(_crack, _crack_size)</code>	void	It is set whether an artificial hole is inserted into the network at the boundary and its size is specified. <i>This function is not used in the current version of the program.</i>
<code>set_crack_center(_crack, _crack_size)</code>	void	It is set whether an artificial hole is inserted into the network in the center and its size is specified. <i>This function is not used in the current version of the program.</i>
<code>set_crystallization(_crystallizing)</code>	void	The network is defined to be crystallizing or non-crystallizing.
<code>set_cut_off(_cutoff)</code>	void	The cut-off radius for the interaction of semi-crystalline links and the "skin"-radius for the corresponding neighbor list is defined.

Continued on next page

Table 4 - continued from previous page

Function	Type	Description
<code>set_data_saving(_save, _local_link_data)</code>	void	It is set whether positions of the nodes, the size of the simulation box and information about the link is saved in a file by calling <code>save_data()</code> .
<code>set_defects(_defects)</code>	void	The fraction of links is defined which will be assigned to be non-crystallizable in a crystallizable network. <i>This function is not used in the current version of the program.</i>
<code>set_eta(_eta)</code>	void	The strength η of the interaction of semi-crystalline links is defined.
<code>set_filler(filled, _filler_frac, _filler_bonds)</code>	void	It is set whether the network is filled. The fraction of nodes which will be of type "filler" is defined. The number of bonds of a filler node can be specified. <i>This function is not applied in the current version of the program.</i>
<code>set_filler_nodes()</code>	void	The type "filler" is assigned to a given fraction of nodes.
<code>set_filler_links()</code>	void	This function sets up filler-filler and polymer-filler bonds, in particular their equilibrium end-to-end distances are defined.
<code>set_force_hist(_make_force_hist, _force_hist_lambda)</code>	void	It is set whether the data for obtaining a free energy density histogram is saved to a file and at which stretches the data is saved.
<code>set_initial_links_morphology()</code>	void	This function sets the number of Kuhn segments of each link and computes the midpoint. Defects, i.e. individual non-crystallizing links, can be set, but <i>this is not used in the current version of the program</i> . Filler-filler and polymer-filler bonds are not completely set up here.
<code>set_kappa(_kappa)</code>	void	The perturbation factor κ is defined.
<code>set_links()</code>	void	This function sets the number of Kuhn segments of each link and computes the midpoint. Defects, i.e. individual non-crystallizing links, are set. This function is only used for unfilled networks.
<code>set_morphology_generator(_use_morphology_generator, _filler_frac, _MC_attempts)</code>	void	It is set whether the MG is applied for the distribution of filler and the fraction of filler is defined. The number of MC attempts per node is specified.
<code>set_morphology_neighbors()</code>	void	A neighbor list, which is used in the MG, is set up for each node.

Continued on next page

Table 4 - continued from previous page

Function	Type	Description
<code>set_nodes_neighbor_list_filler()</code>	<code>void</code>	This function is called in <code>nodes_neighbor_list()</code> if a filled network is considered.
<code>set_rupture(_rupture, _rupture_energy, _max_cryst_rupt)</code>	<code>void</code>	It is defined whether links can break and the parameters for the rupture criterion are specified. <i>The crystallinity is not applied in the current version of the program.</i>
<code>stress()</code>	<code>void</code>	The engineering stress is computed.
<code>stretching(_strain, _previous_strain)</code>	<code>void</code>	The simulation box is deformed to the stretch <code>_strain</code> and the positions of the nodes are adjusted accordingly.

3 Additional Header Files

The file `randomnumbers.h` includes a random number generator (RNG) according to p. 282 in the book "W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*, 2nd ed., Cambridge University Press; 2002". This RNG is used in the MG and for the perturbation of the initially homogeneous lattice of the network.

The file `ucomm.hpp` is used for input of some parameters in the file `main.cpp`.

In `helper.hpp`, the RNG is implemented which is applied to randomly draw the number of Kuhn segments for each link from a Gaussian distribution. The approximation of the exponential function used for the computation of the interaction of semi-crystalline links.