OCTA

Integrated Simulator for Soft Materials

for OCTA

version 1.2

User's manual

OCTA Users Group

Mar. 1, 2013

Authors

All Chapters: Yuichi Masubuchi
Sections relating to OCTA: Taku Ozawa, Yuzo Nishio, and Hideyo Yoshida

Program Developers

Main Program: Yuichi Masubuchi
UDF Converter: Yuzo Nishio and Hideyo Yoshida

Acknowledgements

The theory for this specific simulator has been formulated by a joint study with Prof. Giuseppe Marrucci and Prof. Giovanni Ianniruberto of the Department of Chemical Engineering at The University of Naples Federico II, Italy, and Prof. Francesco Greco of CNR, Italy. We have useful suggestions from Prof. Masao Doi of Tokyo University regarding the theory of polymer blends.

NAPLES utilizes the outcomes from: “Control of Melt Structures by Ultra High Speed Simulator for an Entangled Polymer System” from an Industrial Technology Research Grant Program, in fiscal year of 2001 – 2003 funded by the New Energy and Industrial Technology Development Organization (NEDO); “A Study on a Novel High Speed Calculation Method for Polymer Dynamics in Entangled Polymer Blends” from Grant-in-Aid Scientific Basic Research (C2), in fiscal year of 2003 – 2004 funded by the Japan Society for the Promotion of Science; “Analog-and-Digital United Nano-Simulation of Polymers” in PRESTO program in fiscal year of 2004 – 2007 funded by the Japan Science and Technology Agency (JST); “Multi-scale simulations for Soft Materials” in CREST in fiscal year of 2007 – 2010 funded by the Japan Science and Technology Agency (JST); “Molecular Models for Branched Polymers” from Grant-in-Aid Scientific Basic Research (B), in fiscal 2008 – 2010 funded by the Japan Society for the Promotion of Science; and “Relaxation of Polymers under Fast Flows” from Grant-in-Aid Scientific Basic Research (B), in fiscal 2011 – 2013 funded by the Japan Society for the Promotion of Science.

The gt2et.exe, gt2gw.exe, smooth.exe and gwsmooth.exe, which are useful for the analysis of NAPLES results, have been developed by Prof. Jun-ichi Takimoto of Yamagata University, and they are included in the OCTA-PASTA. We would like to express our gratitude to Prof. Takimoto for allowing us to use his codes.

Program development of the OCTA version has been conducted in: the university collaboration with the industrial science and technology frontier project, “Research and Development of the Platform for Designing High Functional Materials”, commonly referred to as the “Doi Project”. It was funded and assisted by the Ministry of Economy, Trade and Industry, entrusted by the New Energy and Industrial Technology Development Organization (NEDO) to the (General Incorporated Foundation) Japan Chemical Innovation and Inspection Institute; and the OCTA Users Group after the project completed.

Chapter 1 What is NAPLES?

NAPLES is a simulation code for entangled polymer dynamics and rheology, and the name actually stands for “New Algorism for Polymeric Liquids Entangled and Strained”. The code has been developed by Yuichi Masubuchi in collaboration with Prof. Giuseppe Marrucci, Prof. Giovanni Ianniruberto and Prof Francesco Greco at the University of Naples Federico II.

NAPLES is located in a niche between conventional molecular dynamics simulations and molecular constitutive equations[1, 2]. Prediction of polymer dynamics and rheology from molecular details is of industrial significance. Due to the slow nature of entangled polymer dynamics, conventional molecular dynamics simulations are not practically applicable yet. The constitutive equations based on the molecular description, such as tube models, have been thus attempted and have attained remarkable success. However, applicable systems for the theories are still limited, mainly due to the single chain description and mean-field assumption of the multi-chain effects. NAPLES has been developed to take advantages from the two approaches above. Namely, the code is based on the coarse-grained description essentially common to the tube theories, and it calculates multi-chain dynamics like molecular simulations. For further details on the theoretical background, refer to Chapter 6.

Note that no tutorial is provided in this manual, and the other textbook[3] is available for such a purpose.

Possible applications are listed below.

Presumed Applications

- Molecular analysis of entangled polymer dynamics
- Prediction of effects of molecular weight, molecular weight distribution, long chain branching, blending and copolymerization of polymers on polymer dynamics and rheology
- Prediction of polymer dynamics and rheology under fast flows of various deformation modes.
- ...

For convenience, a typical workflow is summarized below with corresponding Chapters for details.

Workflow for the simulation

1. Before executing the code, an input parameter file is necessary. See Chapter 4.
2. naplesOCTA(.exe) is executed with a given input file. Details are given in Chapter 2.
3. The calculation results will be recorded in various output files during the calculation, as described in Chapter 5.
4. For the analysis of the results, refer to Chapter 3, as well as Chapter 2 for the command references.

Chapter 2 Installation and Command References

2.1 Installation

2.1.1 About the OCTA version of NAPLES

The OCTA version of NAPLES includes the Windows and X86-Linux versions of the executable modules and analysis tools. The source code for the executable modules is not being open to the public currently. Use of Linux through a RedHat or Fedora Core distribution is recommended. If there are any problems with these distributions, let us know.

2.1.2 Installation Procedures

The straightforward and recommended installation method is the use of OCTA installer. Then, the following files are installed to the OCTA installation directory. When the installer is not used, the following files must be placed in a directory that is found in the default program search path. Note that the analysis tool includes some codes (created by Prof. Jun-ichi Takimoto at Yamagata University) contained in the PASTA engine for OCTA.

1. gt2gw(.exe): This module converts the linear relaxation modulus to G' and G" curves (being equivalent to gt2gw(.exe) in OCTA-PASTA).
2. gt2et(.exe): This module converts the linear relaxation modulus to linear shear stress growth curves (being equivalent to gt2et(.exe) in OCTA-PASTA).
3. smooth(.exe): This module calculates the logarithmic moving-average for a sequence (being equivalent to smooth(.exe) in OCTA-PASTA).
4. gwsmooth(.exe): This module calculates the logarithmic moving-average for G' and G" sequences obtained from gt2gw.exe.
5. corr(.exe): This module calculates the autocorrelation function of a sequence.
6. getgval(.exe): This module obtains G' and G" from the dynamic shear mode results.
7. naplesOCTA(.exe): This file is the NAPLES-executable module.
8. naplesgwOCTA(.exe): This module attains dynamic shear mode calculations.
9. naplesOCTA.sh (.bat in Windows version): This batch file kicks NAPLES with a given input UDF file.
10. naplesgwOCTA.sh (.bat in Windows version): This batch file realizes dynamic shear model calculatiions with a given input UDF file.
11. naples_input_converter(.exe): This module converts an input UDF to the input data format required for naplesOCTA.
12. naples.udf: This file is necessary for UDF format data.

In addition to the files above, some action files for analyzing calculation results (functions for analyzing stress, for converting stress to actual units, and for plotting stress) are presented, though not explicitly presented above.

2.2 Command Reference for naplesOCTA

2.2.1 Starting naplesOCTA(.exe)

The following section will cover the use of the NAPLES-specific file format, not the UDF format, as the Input file. For the use of UDF formatted input file, see Chapter 4. To start the code, launch the terminal in the case of UNIX, or cygwin or the DOS prompt in the case of Windows (when cygwin is installed, there is a shortcut on the desktop). Then start naplesOCTA.exe as shown below, in the directory where the input parameter file is placed.

Starting naplesOCTA(.exe)

```
>naplesOCTA(.exe)   [ input parameter file]
```

If the input parameter file is not specified, naplesOCTA.exe will use input.npls in the current folder. If input.npls does not exist, naplesOCTA.exe will start with the initial conditions which are embedded into the code.

Two files, movelog.npls and prmcheck.npls, are created in the folder where NAPLES is started, if it has started normally. The contents of these two files may be checked with a text editor such as notepad. An example of the contents for movelog.npls is shown below

.

Checking movelog.npls (for cygwin and UNIX)

```
>cat movelog.npls              "Display the contents of movelog.npls"
-------------------------
naplesOCTA(.exe) is started (NAPLESrev ) with parameters in input.npls Thu Mar 17 18:32:40
200 Initialization started. Thu Mar 17 18:32:40 2005
Main calculation started. 0 Thu Mar 17 18:32:46 2005
EQUILIBRIUM_CALC Thu Mar 17 18:32:46 2005
```

The results are stored in the stress output file, measure.npls. The end section of the measure.npls file in cygwin or Linux could be shown as follows. Details of the output files can be found in Chapter 5. Other output files may be created with respect to the parameters embedded in the input file.

Checking measure.npls (for cygwin and UNIX)

```
>tail measure.npls              "Display the end section of measure.npls"
 0     0.00e+00        5.519445e-01    1.430961e+00    1.137091e+00    1.193193e+00
 1     0.00e+00        1.126641e-01    1.273881e+00    1.186699e+00    1.205338e+00
 2     0.00e+00        8.506818e-02    1.227362e+00    1.163581e+00    1.177303e+00
 3     0.00e+00        4.590036e-02    1.193680e+00    1.152211e+00    1.175619e+00
 4     0.00e+00        3.322798e-02    1.158326e+00    1.172665e+00    1.141329e+00
```

2.2.2 Stopping naplesOCTA(.exe)

There are two ways to stop a calculation in progress. If the current configuration needs to be stored in order to restart the calculation later, perform the termnation procedure described below. If there is no need to restart the calculation, the Windows Task Manager, or the cygwin or Linux kill command, may be used as well. For the finalization with generating a configuration file for restarting, find the ctrl.npls in the directory where naplesOCTA(.exe) was launched (the directory where measure.npls file, etc., are also placed). The content of ctrl.npls should be as follows.

Content of ctrl.npls during execution of naplesOCTA(.exe)

```
RUN
```

To terminate the execution, rewrite the content as follows and save the file. naplesOCTA reads ctrl.npls at constant time intervals, and it will eventually rewrite the content of the configuration-save file, finalconf.npls, and then stop the calculations. To restart a halted calculation, refer to Section 2.2.3.

Changing the ctrl.npls contents to stop naplesOCTA(.exe)

```
STOP (or KILL)
```

2.2.3 Restarting naplesOCTA(.exe)

naplesOCTA could be restarted from a certain saved configuration. For the restarting, a damped configuration file, and a restarting command in the input file, are required.

The required damped configuration file is named as finalconf.npls, and the file is automatically generated with periodic manner during the code running, as well as at the termination of running. This file is placed in the folder where NAPLES was launched, and it should be in the folder at which the restarting run will be performed. Note that finaconf.npls is periodically overwritten so that the original file is suggested to be stored separately.

Unless a specific command for the restarting calculation is written in the input file, the damped configuration file is not read into the code. For the restarting, the statement shown below is necessary in the input parameter file. When the statement exists in the input file, NAPLES searches for finalconf.npls. If the finalconf.npls file does not exist in the same folder with naplesOCTA, the code outputs an error message to the movelog.npls file and stops the execution.

Restarting command to be placed in input file

```
calculation_mode=restart
```

2.3 Using UDF with naplesOCTA

For the code running with respect to the parameters stored in UDF-formatted input files rather than the original NAPLES format files, the conversion of the input file is required. A straightforward way is the use of naplesOCTA.bat (.sh in Linux version) instead of naplesOCTA(.exe) for the automatic file conversion, as shown below.

Executing a calculation when using a UDF file as Input

```
>cd d:/NAPLES                                   "Move to NAPLES folder"
>naplesOCTA.bat(.sh in the Linux version)   [input UDF file]
```

2.4 Command Reference for Analysis

This section describes the group of files for analyzing calculation results. When UDF is used for the input file format, these functions can be executed using the UDF action functions. For the specific details for the analysis actions, refer to Chapter 3.

2.4.1 gt2et(.exe)

By this command the Linear stress growth function $\eta^+(t)$ is calculated from the linear relaxation modulus G(t).

$$\eta^+(t) = \int_0^t G(t')dt' \tag{2.1}$$

The following line shows how to use gt2et.exe from the command line.

How to use gt2et.exe

```
gt2et.exe [-s dt] < [input file] > [output file]
```

The time step is specified by dt, which is set to 1.0 unless given. The input file format should be as follows.

Input file format for gt2et.exe.

$G(0)$
$G(dt)$
$G(2dt)$
...

$G(t)$ here represents the linear relaxation modulus. The time and $\eta+(t)$ at that time are output as tab separated variables on each line of the output file. This command is the same with that in OCTA-PASTA.

2.4.2 gt2gw(.exe)

By this command the complex viscosity $\eta*(\omega)$, and the complex elastic modulus $G*(\omega)$, are calculated from the linear relaxation modulus $G(t)$.

$$\eta^*(\omega) = \int_0^\infty G(t)\exp(i\omega t)dt \tag{2.2}$$

$$G^*(\omega) = i\omega\int_0^\infty G(t)\exp(i\omega t)dt \tag{2.3}$$

The following line shows how to use gt2gw.exe from the command line.

How to use gt2gw.exe

```
gt2gw.exe [-n m] [-s dt] [-e] < [input file] > [output file]
```

The argument m here leads to the number of data in the form $N = 2^m$ used for the Fourier transformation. The default value of m is 16, and M must be selected so as to satisfy $2^m > 4M$, where M represents the number of data actually given for the analysis. The argument dt represents the time step in the input file, and it is set to 1.0 unless specified. When the -e option is given, the viscosity is calculated according to eq 2.2, whereas the modulus is calculated as long as the option is not specified. The format for the input file should be the same as for gt2et, and is as follows.

Input file format for gt2gw.exe

$G(0)$
$G(dt)$
$G(2dt)$
...

On the other hand, in the output file, the frequency, and the viscosity or elastic modulus, are written with a step of $d\omega = 2\pi(Ndt)$ as,

Output file format for gt2gw.exe (in the case of G' and G")

0	$G'(0)$	$G"(0)$
$d\omega$	$G'(d\omega)$	$G"(d\omega)$
$2d\omega$	$G'(2d\omega)$	$G"(2d\omega)$
$3d\omega$	$G'(3d\omega)$	$G"(3d\omega)$
$\cdots$	$\cdots$	$\cdots$
$N/2d\omega$	$G'(N/2d\omega)$	$G"(N/2d\omega)$

This command is the same with that in OCTA-PASTA.

2.4.3 smooth(.exe)

By this command, a given data sequence is smoothed with respect to the weighted moving average method, so-called Savitzky-Golay method. Specifically, this command attains the averaring in logarithmic scale, which is often used for rheological measures. The following line shows how to use smooth.exe from command line.

How to use smooth.exe

```
smooth.exe [-n ndiv] [-r r] [-x nmax] [-s skip] -2 [input file] > [output file]
```

Here, the parameter ndiv, following –n option, represents the number of points output for one order of magnitude in the logarithmic scale, and it is set to 20 unless specified. The parameter r, following –r option, is the range of data for taking the moving average. Namely, assuming the average in logarithmic manner, for a specific value x at a certain time t, the average is calculated in the range from x(t/2) to x(2t). The value of r is set to 2.0 as default. The parameter nmax, following –x option, specifies the number of processed data points, and is set to 1000 unless specified. The value of skip, following –s option, is the number of data points for which the data processing will not be made and the raw data will appear as the output. The option -2 is for the calculation using the second order Savitzky-Golay smoothing.

The input file for smooth.exe must be formatted as follows.

Input file format for smooth.exe

t_0	$v(t_0)$
t_1	$v(t_1)$
t_2	$v(t_3)$
$\cdots$	$\cdots$

Here, t_0, t_1, and t_2 must satisfy $t_0 < t_1 < t_2$; however, the intervals between t_0 and t_1 and between t_1 and t_2 are not necessarily equal (moreover, t, which is written here for the sake of convenience, may be ω.) The output is formatted as follows.

Output file format for smooth.exe

T_0	$\tilde{v}(T_0)$
T_1	$\tilde{v}(T_1)$
T_2	$\tilde{v}(T_3)$
$\cdots$	$\cdots$

This command is the same with that in OCTA-PASTA.

2.4.4 gwsmooth(.exe)

Files with a three-row (two rows of data for one row of frequency or time) format, such as the file in which the complex modulus is recorded, can be smoothed by using gwsmooth(.exe).

How to use gwsmooth.exe

```
gwsmooth.exe [input file] > [output file]
```

The details regarding smoothing are similar to those for smooth.exe in Section 2.4.3. Both input and output files have the following format.

Input/output file format for gwsmooth.exe

$$
\begin{array}{lll}
t_0 & v_1(t_0) & v_2(t_0) \\
t_1 & v_1(t_1) & v_2(t_1) \\
t_2 & v_1(t_3) & v_2(t_3) \\
\cdots & \cdots &
\end{array}
$$

Here, v_1 and v_2 represent data.

2.4.5 corr(.exe)

The autocorrelation function of a sequence of data is calculated by corr.exe.

$$C(\tau) = \frac{\langle \sigma(t)\sigma(t+\tau) \rangle_t}{\langle \sigma(t)^2 \rangle_t} \tag{2.4}$$

Here, $\langle \cdots \rangle$ represents time average. The following line shows how to use corr.exe.

How to use corr.exe

```
corr.exe [-s skip] [input file] > [output file]
```

Here, the value skip following the option –r gives the number of data points without calculating the correlation. The value is set to 100 unless specified. The input file must be formatted as follows, as with gt2et.exe and gt2gw.exe.

Input file format for corr.exe

$$
\begin{array}{l}
G(0) \\
G(dt) \\
G(2dt) \\
\ldots\ G(\ \mathrm{datanum}\ dt)
\end{array}
$$

The number of steps, and $C(\tau)$ in formula 2.4, are output with a time step of dt.

2.4.6 naplesgwOCTA(.exe)

This module automates the application of a dynamic (sinusoidal) shear strain in a manner similar to experiments, while changing the frequency. The obtained results are analyzed by using getggval.exe (Section 2.4.8) to finally obtain the complex modulus $G*(\omega)$ (file name gwdirect.npls: See Section 5.10).

How to use naplesgwOCTA(.exe)

```
naplesgwOCTA(.exe) ([-options] values) npls file]
        -wmax [the maximum frequency to be scanned: default 1.0]
        -wmin [the minimum frequency to be scanned: default 0.001]
        -div [number of divisions for one decade: default 10]
        -amp [strain amplitude: default 0.3]
        -cycle [strain cycle to be scanned: default 3.0]
        -input [input file (.npls format) : default input.npls]
        -config [configuration file to be read as the initial configuration. ]
        -prerun [pre-thermalization steps if the equilibrated configuration does not
                exist.: default 1000 ]
        -output [output file for G' G'' against w: default gwdirect.npls]
        -engine [engine of naples for calculation: default naplescalc.out]
```

2.4.7 Using UDF with naplesgwOCTA(.exe)

naplesgwOCTA.exe could be used with an input file in UDF format via naplesgwOCTA.bat, which converts the UDF for the use of naplesgwOCTA.exe, and then it kicks the code running. For this sake, the naples_input_converter(.exe), naplesgwOCTA(.exe), naplesOCTA(.exe), and getggval(.exe) files should be located in the default program search path. The options are in accordance with Section 2.4.6.

How to use naplesgw with an UDF

`naplesgwOCTA.bat` (`.sh` in Linux version) [name of input UDF file] [argument to naplesgwOCTA]

2.4.8 getggval(.exe)

This module obtains G' and G'' from the results obtained under dynamic shear deformations, which are attained via naplesgwOCTA modules explained in sections 2.4.6 and 2.4.7 above. The following line shows how to use getggval(.exe).

How to use getggval.exe

```
> getggval.exe [frequency value] [strain amplitude value] measure.npls > [frequency] [G'] [G"]
```

The G' and G'' to be obtained will only give a pair of values at the specified frequency. When a dynamic viscoelasticity curve is necessary, first repeatedly perform the calculation as the frequency changes by using naplesgwOCTA.

Chapter 3 Rheology Calculations and Analysis

NAPLES calculates the motion of molecules under given flows or deformations, and obtains the stress from the shape of the molecules. This chapter gives examples for the analysis of rheological measures under typical flows and deformations.

3.1 Stress Relaxation and Relaxation Modulus under Step Deformations

3.1.1 Performing Simulation

Make the simulation with an appropriate input file in which details of the system and the parameters describing the deformation are given. An example of the input file for the step-shear calculation is given below. Explanations for the parameters are given in Chapter 4. See sections 2.2 and 2.3 to execute the simulation.

Example of input file for step deformation

```
cell_size=12
seed=87911
quit_dynamics_count=4000
component_number=1
molecule_spec1=linear
Z1=10
fraction1=1.0
flow_type=step_shear
initial_strain=1.0
pre_reptation_step=100
```

3.1.2 Extracting Stress

The simulation records the time development of stress in measure.npls file. Fig. 3.1 shows an example of the shear stress (3rd row of measure.npls) plotted against time (1st row of the file).

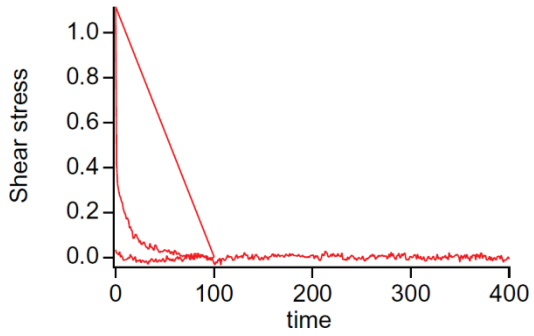

Fig. 3.1: Example of stress output under step shear

In calculating step stress relaxation, there is usually an equilibration time as specified by pre_reptation_step, and thereafter the stress gradually decays from the time at which the deformation is applied. NAPLES automatically resets the time to 0 when a step deformation is applied, and therefore the data during equilibration should be removed with a spreadsheet software, or something similar. For this specific example, the setting for the equilibration is given as pre_reptation_step=100, so that the data during equilibration (from t=0 to 100) has been removed to obtain Fig. 3.2.

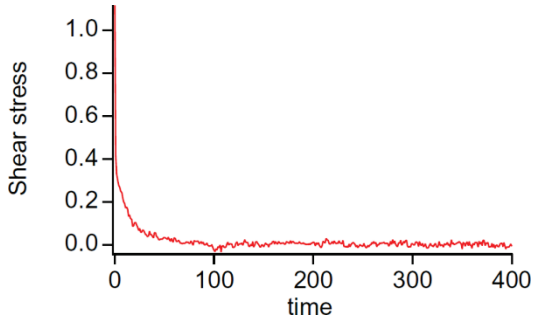

Fig. 3.2: Example of stress relaxation in step shear: the section for equilibration has been removed

3.1.3 Smoothing Data

Due to the stochastic nature of the simulation, the obtained stress fluctuates in time, and thus, smoothing of data (moving-time average) is useful. As given in section 2.4.3, the module for data smoothing is available in the package, and the module requires the input specifically formatted. More specifically, the module, smooth.exe (Section 2.4.3), requires an input file formatted in two rows, for which the first and second rows stand for the time and measure, respectively. An example of the data extraction from measure.npls is given below.

Extracting time and shear stress using awk

```
>awk␣'{printf("%e␣%e␣\n",$1,$3)}'␣<␣measure.npls␣>␣[file for extracted data]
```

Here, $1 represents the time row (first row). In this example, the time row and the shear stress in $3 (third row) are extracted, and the values in $3 are appropriately changed to $4, $5, etc., for the case of normal stress. See section 5.3 for further detail of measure.npls.

As long as the stress is appropriately extracted from measure.npls, it can be smoothed by smooth.exe.

Smoothing data using smooth.exe

```
smooth.exe␣[file to be smoothed]␣>␣[result file]
```

Fig. 3.3 shows the results together with the data before smoothing.

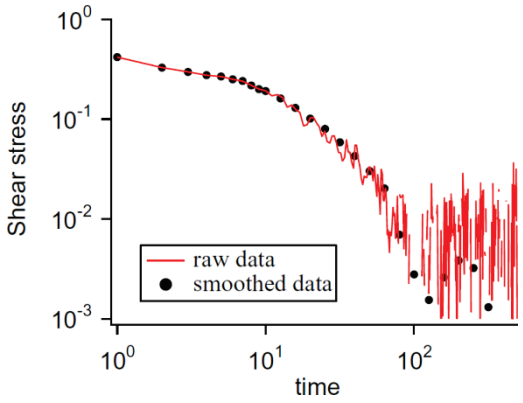

Fig. 3.3: Example of stress relaxation in step shear: result of smoothing (displayed in double logarithmic plot)

3.1.4 Converting Stress to Modulus

The modulus G is determined in the following relation with stress σ and strain γ.

$$G = \frac{\sigma}{\gamma} \tag{3.1}$$

Fig 3.4 shows the relaxation modulus $G(\gamma, t)$ obtained from the stress calculated under various initial strain.

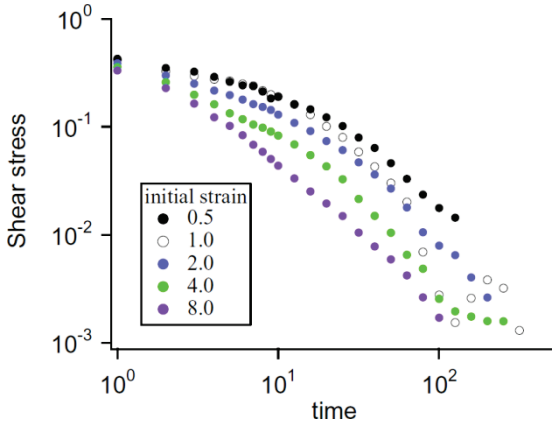

Fig. 3.4: Example of a stress relaxation curve

In principle, the linear relaxation modulus G(t) can be obtained with a small strain. However, this strategy is not practical due to poor signal/noise ratio under small strains. For the linear relaxation modulus, the way recommended is the use of Green-Kubo formula. See section 3.2.

3.2 Linear Relaxation Modulus from Equilibrium Simulations

Owing to the linear response theory, the linear relaxation modulus corresponds to the stress auto-correlation function under equilibrium. This section explains this strategy.

3.2.1 Performing Simulation

For the analysis, fluctuation of stress (time-development of stress) under equilibrium is necessary. An example of the input file for the simulation is shown below. Apart from the description of the system, the important parameters are flow_type (must be set to equilibrium) and quit_dynamics_count. In particular, the latter should be sufficiently (at least 10 times) longer than the longest relaxation time of the system. See section 3.8.1 for estimation of the longest relaxation time. If the estimation is of difficulty, a suggested value of quit_dynamics_count is 100,000, for which the simulation takes roughly one night, if the cell_size is set to 10.

Example of input file for equilibrium calculation

```
cell_size=8
seed=87911
quit_dynamics_count=100000
component_number=1
molecule_spec1=linear
Z1=10
fraction1=1.0
flow_type=equilibrium
```

3.2.2 Extracting Stress Fluctuation

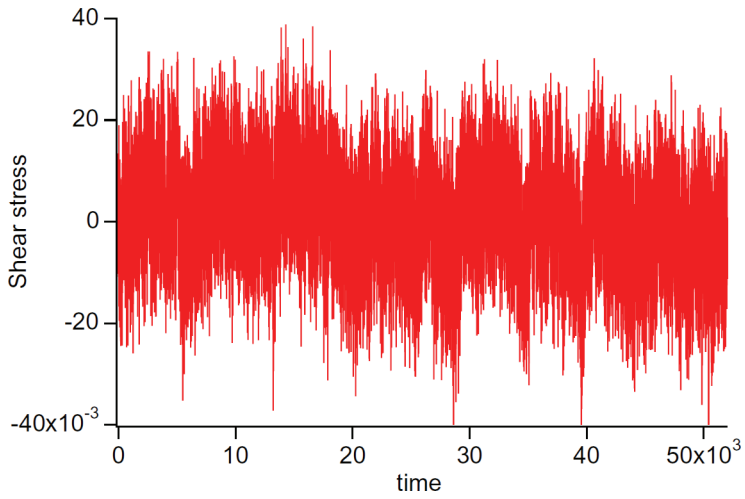

Fig. 3.5: Fluctuation of shear stress under equilibrium

During the equilibrium simulation, the stress fluctuates in time as shown in Fig 3.5, and this fluctuation will be used for the calculation of G(t). For this sake, the recorded stress fluctuation is extracted from the output file measure.npls. This operation can be attained using awk under the UNIX or cygwin environments as shown below. In this example, the shear stress in $3 (3rd row) is extracted.

Extracting shear stress using awk

```
>awk␣'{printf("%e␣\n",$3)}'␣<␣measure.npls␣>␣ [name of file for saving the stress]
```

3.2.3 Calculating Correlation Function

Owing to the fluctuation-dissipation theorem, the linear relaxation modulus corresponds to the auto-correlation function of stress. The auto-correlation function for a given series of valuable can be calculated by the module corr.exe as described in section 2.4.5. Note that the correlation must be calculated for equilibrium state, and the stress data during the system equilibration should be excluded. Namely, in the use of corr.exe, the option –s and the parameter "skip" are of importance. Because the equilibration takes a certain time comparable to the longest relaxation time of the system, the value of "skip" must be larger than that. For the estimation of the longest relaxation time of the system, see section 3.8.1. If the estimation is not available, a reasonable way is to set the parameter to the half of total simulation steps. (Namely, if the simulation was performed for 100,000 steps, set "skip" at 50,000).

An example of calculation for G(t) is shown below. Assume that the simulation result, measure.npls, is located in a directory named "Data", and the modules corr.exe and smooth.exe are placed in a directory "NAPLES/Tools/". Both directories are in D drive. The operation for calculation can be attained as follows. In this example, the raw data for the stress fluctuation is in the file s12, the correlation function thereof is output to s12corr, and the smoothed data is written in s12tsm.

Example of calculating G(t)

```
>cd␣d:/Data                "Move to data folder"
>gawk␣'{printf("%e␣%e␣\n",$1,$3)}'␣<␣measure.npls␣>␣s12 "Extract shear stress data"
>d:/NAPLES/Tools/corr.exe␣-s␣1000␣s12␣>␣s12corr   "Calculate the correlation function"
>d:/NAPLES/Tools/smooth.exe␣s12corr␣>␣s12tsm       "Smoothing"
```

Figure 3.6 shows the calculated G(t) from the equilibrium calculation (solid curve) in comparison to the relaxation modulus obtained from the simulations with step shear deformations. Although the accuracy is still inadequate in the long-time range due to the poor statistics, the results of the step shear deformation and the results from determining the correlation function are essentially equal, as expected. For further accuracy, see chapter 7.

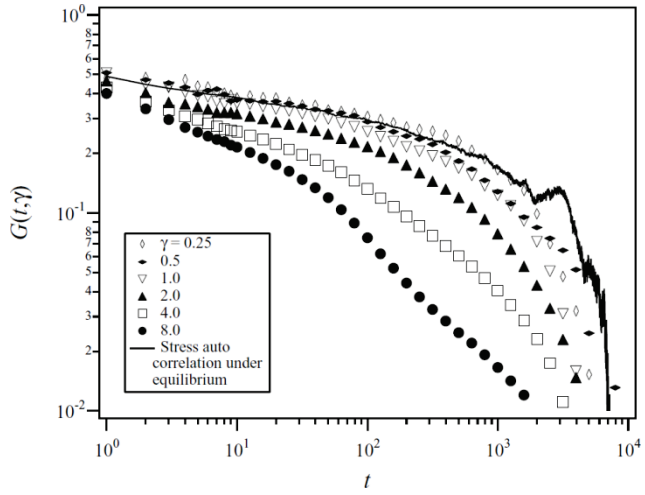

Fig. 3.6: Comparison of the correlation function obtained under equilibrium, with the relaxation modulus obtained under step shear deformations

3.3 Complex Relaxation Modulus G' and G"

Essentially there are two routes for determining G' (storage modulus) and G" (loss modulus). The options are converting G(t) and obtaining G' and G" directly with dynamic shear deformations as described in this section.

3.3.1 Calculating G' and G" from G(t)

Owing to the definition, G' and G" are obtained from G(t) via Fourier transformation. The tool for this operation is gt2gw.exe, and the usage is shown in section 2.4.2. The obtained G' and G" could be smoothed by gwsmooth.exe module (see section 2.4.4).

3.3.2 Example of Calculation from Linear Relaxation Modulus

An example of an actual calculation is shown here. The input file for this example is shown below.

Example of input file

```
cell_size=10
seed=845
quit_dynamics_count=50000
flow_type=equilibrium
molecule_spec1=linear
Z1=8
```

After the simulation, G(t) was obtained following the procedure described in section 3.1. The result is shown in Fig. 3.7. For this G(t), G' and G" were obtained via gt2gw.exe, and they are shown in Fig 3.8. Finally, the G' and G" were smoothed via gwsmooth.exe, and the results are shown in Fig 3.9. Note that there are a few software packages for the conversion from G(t) to G'G", such as reptate.com and i-Rheo. These external software packages may give better results for the conversion.

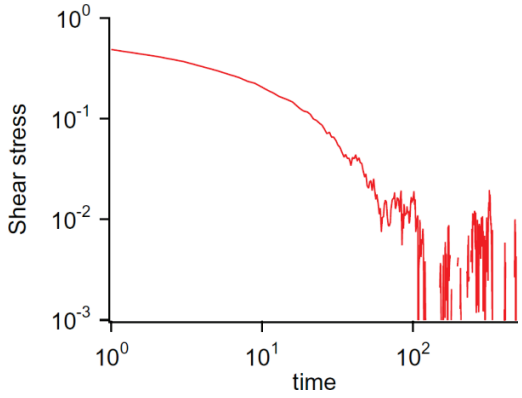

Fig. 3.7: Linear relaxation modulus

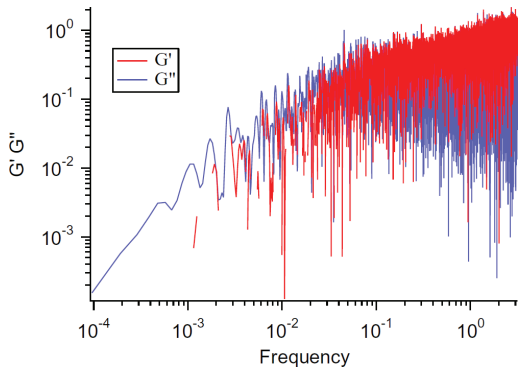

Fig. 3.8: G’ and G”

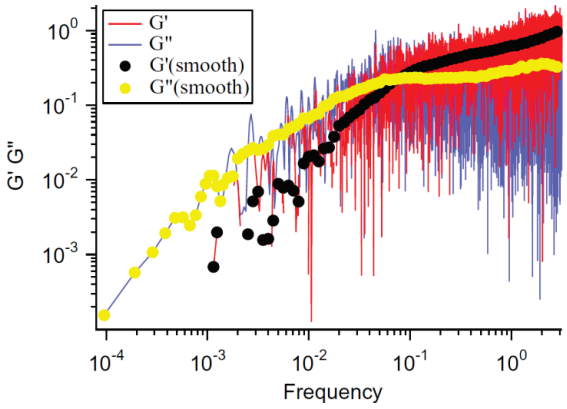

Fig. 3.9: G' and G" with smoothing

3.3.3 Determining G' and G" from Dynamic Viscoelastic Test

Just as in experiments, oscillatory deformations can be applied to the system in the simulation, and G' and G" can be obtained from the stress response. Fig. 3.10 shows an example of the stress response to a oscillatory strain.

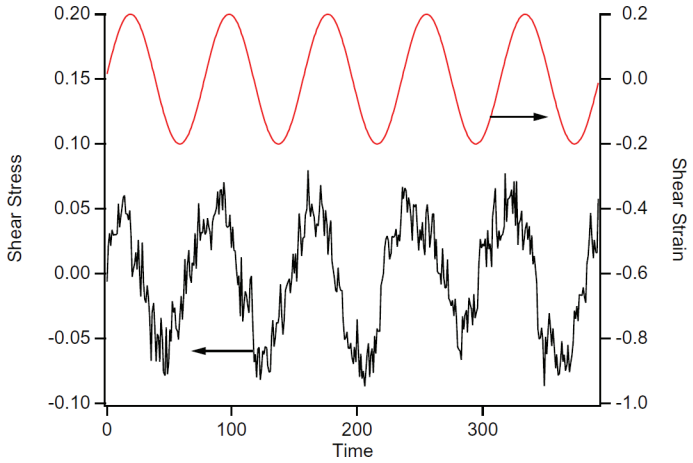

Fig. 3.10: Stress response to oscillatory strain

As made by rheometers in experiments, the stress response is fitted to the functional form shown in eq 3.2.

$$\sigma(t) = \sigma'(\omega)\sin\omega t + \sigma''(\omega)\cos\omega t \tag{3.2}$$

From the obtained values of σ' and σ'', G′ and G″ are determined as $G'(\omega) = \sigma'/\gamma_0$ and $G'(\omega) = \sigma''/\gamma_0$ respectively, with the strain amplitude γ_0. For this fitting operation, the module getggval(.exe), is used as follows.

Calculating *G'* and *G″* from measure.npls

```
>getggval.exe [frequency] [strain amplitude] measure.npls
>[frequency] [G′ value] [G″ value]
```

In the experiments, G′ and G″ are obtained as functions of frequency by a series of measurements in which the frequency is automatically scanned in a given range. Such an operation can be attained by the module naplesgwOCTA(.exe). See section 2.4.6 for details. The results will be stored in gwdirect.npls file (see Section 5.10).

3.4 Stress Growth under Start-up Flows

3.4.1 Performing Simulation

Stress growth and corresponding viscosity growth under fast start-up flows can be simulated by setting the flow conditions in the input file accordingly. An example is shown below. Here, a pom-pom branch polymer melt is exposed to a uniaxial elongational flow. For the details of the flow settings, refer to Section 4.6.5. With such an input file, the simulation can be performed by naplesOCTA.exe as described in Section 2.2.1.

Example of input file

```
cell_size=12
seed=4799
quit_dynamics_count=3000
flow_type=uniaxal_startup
strain_rate=0.001
pre_reptation_step=300
molecule_spec1=pompom
armZ1=3
arm_per_junction1=4
bridgeZ1=10
```

3.4.2 Extracting Elongational Stress

The normal stress are stored from 4th to 6th rows in the measure.npls file. For the case of uniaxial deformations, the elongational stress is usually defined as the difference between the normal stress parallel and perpendicular to the stretching direction. For the uniaxial elongational flows in naplesOCTA.exe, the 4th row is the stress to the stretching direction, and the others are to the perpendicular directions. An example for the operation to extract the uniaxial stress from a given measure.npls file is as follows.

Creating a file containing uniaxial stress growth using awk

```
>awk␣'{printf("%e␣%e␣\n",$1,$4-$5)}'␣<␣measure.npls␣>␣[file for saving the data]
```

3.4.3 Example of Calculation for Stress Growth

Fig. 3.11 shows the uniaxial stress growth obtained in the operations mentioned above. Because the equilibration steps are included in the result file (owing to the input parameter “pre_reptation_step”), such a segment has to be removed. Fig 3.12 shows the data after removing the equilibration part, and the data corresponds to the stress growth under the flow. Note that to maximize the final elongational strain for the calculations, the calculation starts from a unit-cell to which a negative strain has been applied in advance, and which is a cuboid shape other than a cube. In this respect, if the equilibration time specified by pre_reptation_step is insufficient, a negative elongational stress may be detected at a short time. Nevertheless, for a noise reduction purpose, the obtained data may be subjected to the smoothing, as shown in Fig. 3.13.

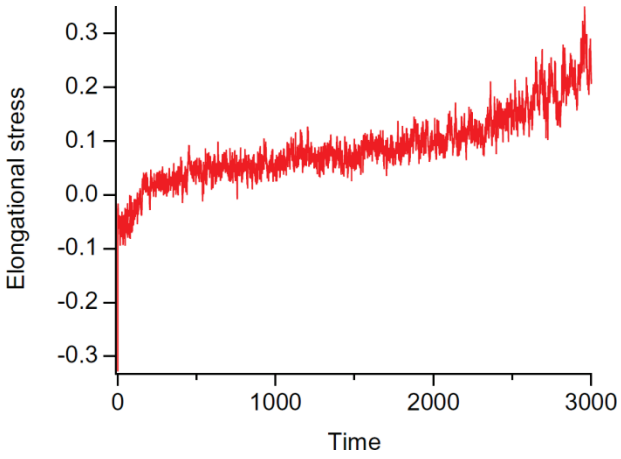

Fig. 3.11: Example of stress growth under uniaxial elongation including equilibration steps

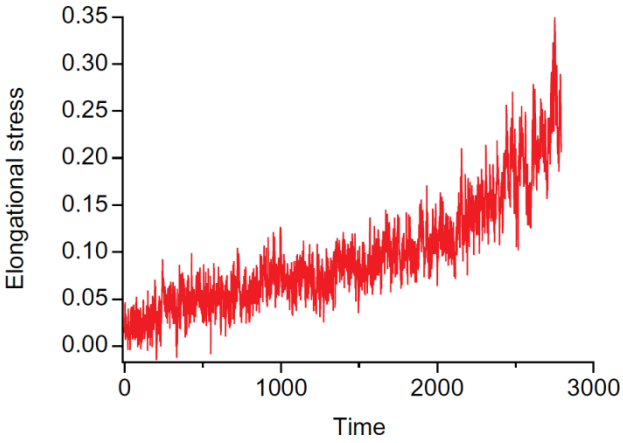

Fig. 3.12: Example of stress growth under uniaxial elongation without equilibration steps

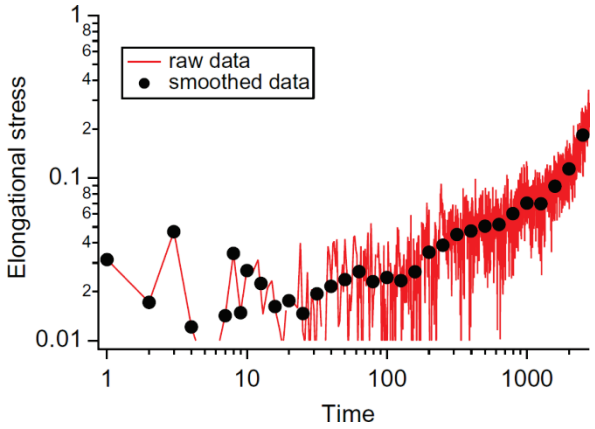

Fig. 3.13: Example of stress growth under uniaxial elongation with smoothing

3.5 Linear Viscosity Growth Curve

As in experiments, obtaining a linear viscosity growth curve is of difficulty by applying flows due to the poor signal/noise ratio under small flow rates. Instead, once G(t) is obtained as explained in Section 3.2, the other linear viscoelastic measures are obtained from G(t) owing to the Boltzmann principle. Indeed, G(t) can be converted to the linear viscosity growth function η(t) by gt2et.exe module as explained in section 2.4.1.

3.6 Converting Stress to Viscosity

Stress obtained under flows can be converted to viscosity as it is defined as stress divided by the strain rate (shear rate or elongation rate) in a flow field. For example, assume that the flow field is set as follows.

Example of flow setting in input.npls

```
flow_type=start_up_shear
strain_rate=0.001
```

Then the shear viscosity is the value dividing the shear stress, which is available in the 3rd row of the resultant measure.npls file, by the shear rate of 0.001. On the other hand, For the case of linear viscosity growth curve, the obtained value is not the stress but the viscosity, and no conversion is necessary.

3.7 Flow Curve

Flow curve is the shear-rate dependence of the steady shear viscosity. Such a relationship could be obtained by an automatic scanning of shear rate as performed experimentally. However, for entangled polymers the auto-scanning procedure is not recommended unless the relaxation time is sufficiently smaller than the scanning time. Nevertheless, for naplesOCTA the scanning procedure is not provided, and the shear-rate dependence of viscosity must be obtained successively.

An example of the calculation is shown below for the input file shown below. The stress growth can be obtained by the simulations explained in Section 3.4, and the stress can be converted to the viscosity as mentioned in Section 3.6. By a series of simulations in which the strain_rate was varied, Fig. 3.14 was obtained. Then, the steady state viscosity was extracted from the result, and plotted against the strain rate as shown in Fig. 3.15, which is the flow curve of this system.

Example of input file

```
cell_size=8
seed=98981
quit_dynamics_count=2000
pre_reptation_step=100
molecule_spec1=linear
Z1=17
flow_type=start_up_shear
strain_rate=0.0013
```

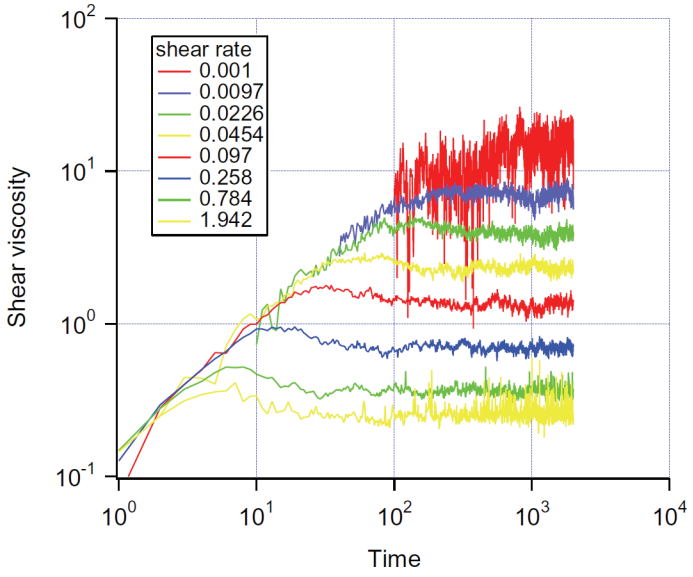

Fig. 3.14: Example of a viscosity growth curves

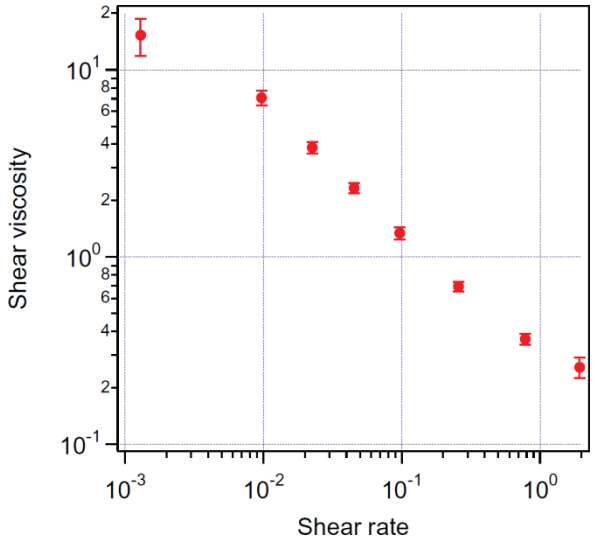

Fig. 3.15: Example of flow curve

3.8 Converting Results to Experimental Values

The obtained rheological measures such as stress, modulus and viscosity are normalized, and they can be converted to experimental values with the unit of time and unit of stress. The latter can be easily determined from the tabulated plateau modulus[4] by the following relation.

$$G_{NAP} = G_0 \times 1.6 \tag{3.5}$$

On the other hand, the unit of time is to be determined via the fitting to the linear viscoelastic data, although it is defined by the following equation.

$$\tau_e = \frac{\zeta a^2}{6kT} \tag{3.3}$$

In this formula, a represents the average length of an entangled tube segment, k the Boltzmann constant, T the temperature, and ζ the friction of the segment. The determination of these parameters is not straightforward.

Chapter 4 Naples Input Files

The input parameters will be described in this section for the NAPLES-specific Input file format. Parameters with almost the same names are used in Input files in UDF format, which is provided in the OCTA version. The relationships between the parameter names used in the UDF format and NAPLES-specific Input files are listed in the last part of this chapter.

4.1 Basic Rules

- Description of comments:
  # or % at the beginning of a line identifies the line as a comment.
- Description of parameters:
  To specify an input parameter in the file, the name of the parameter and a numerical value are connected by "=". Specifically, the basic format is as follows

Basic format for input parameters

```
[name of parameter]=[input value]
```

- Order of parameters:
  Parameters may be specified in any order; however, when a parameter name is specified more than once in the same file, the last value specified for that parameter name is adopted.

4.2 Sample Settings

4.2.1 Number of Chemical Species and Related Parameters

When different chemical species are mixed, the number of chemical species must be specified. Currently, only two kinds of chemical species can be mixed.

Parameter for specifying the number of chemical species

```
chemical_component_number=[number of chemical species, 1 or 2]
```

The interaction between 2 kinds of chemical species is specified by the Flory χ parameter [10].

Interaction parameter between chemical species

```
chi12=[interaction parameter]
```

The molecular weight between entangled points (plateau elastic modulus) is generally different between different polymer species. The ratio of the segment length for the second component to that of the first component is given as follows.

Ratio of the entangled segment length between chemical species

```
bond_ratio_for_chemical[chemical species id]=[ratio of entangled segment length]
```

The friction for the entangled segment is also usually different between different polymer species. The ratio of the coefficients of friction for the second component to that for the first component is given as follows.

Ratio of coefficients of friction per segment between chemical species

```
friction_ratio_for_chemical[chemical species id]=[ratio of coefficient of friction]
```

4.2.2 Finite Extensibility

To introduce the finite extensibility of polymers, the following key is used, otherwise the Gauss spring is assumed.

Introducing finite extensibility

```
FENE_flag=true
```

The maximum elongation ratio must be specified as well.

Specifying the maximum elongation ratio for each chemical species

```
maximum_stretch_for_chemical[chemical species id]=[maximum stretch ratio]
```

The maximum stretch ratio is set to 4 by default, and the value for polystyrene melts is presumed.

4.2.3 Friction-Reduction Effect

To introduce the stretch-orientation-induced reduction of friction (SORF), the following key is used. For details regarding reducing friction, refer to the original paper [5].

Introducing a friction-reducing effect

```
friction_reduction_flag=true
```

4.2.4 Number of Constituents

If the simulation is made for a mixture of polymers with different molecular weights and/or branching structures, the number of components must be specified by the following parameter.

Parameter for specifying constituents

```
component_number=[number of constituents]
```

The volume fraction of each component is specified as follows.

Specifying the volume ratio of constituents

```
fraction[component id]=[volume fraction]
```

In a system with multiple chemical species, it is necessary to specify the chemical species for every component.

Specifying the chemical species for each constituent

```
chemical_spec_for_component[component id]=[chemical species number]
```

The topology and molecular weight of the molecules must also be set for each component. Specifically, the values are set as follows.

Description of the molecular shape for the individual components

```
molecular_spec[component id]=[molecular topology]
Z[component id]=[molecular weight]
....
```

4.2.5 Molecular Topology: Outline

NAPLES utilizes the parameter group shown in Table 4.1 to give the molecular topology and weight within a sample. In the example for specifying each parameter, the numerical value of 1 that follows the name of the parameter represents the component id number.

Table 4.1: Parameters for specifying molecular shape

Parameter name	Definition	Example
molecule_spec	Topology of molecule	molecule_spec1=linear
Z	Molecular weight of a linear polymer	Z1=10.0
armZ	Molecular weight of arm extending from junction	armZ1=5.0
arm_per_junction	Number of arms per junction	arm_per_junction1=5
bridgeZ	Molecular weight between junctions in a pompom or comb molecule	bridgeZ1=6.0
junction_number	Number of junctions per molecule in a comb molecule	junction_number1=10

4.2.6 Linear Polymers

Linear polymer is a polymer without branches, as illustrated in Fig. 4.1.

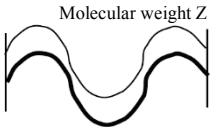

Fig. 4.1: A linear polymer

Molecular weight (parameter name Z) may only be specified for a molecule of this shape.

Example for a linear polymer

```
molecule_spec1=linear
Z1=3.0
```

The parameter Z represents the number of entanglement segments per molecule being obtained by converting the actual molecular weight to the dimensionless molecular weight used in NAPLES. For further details, refer to Section 6.5. When a molecular weight distribution is given, multiple components, each having a different molecular weight, are specified as follows.

Linear polymer system with a molecular weight distribution

```
component_number=10
molecule_spec1=linear
```

```
Z1=3.0
fraction1=0.05
molecule_spec2=linear
Z2=5.0
fraction2=0.1
molecule_spec3=linear
Z3=10.0
fraction3=0.15
molecule_spec4=linear
Z4=20.0
fraction4=0.2
.........
```

4.2.7 Star Polymers

Star polymer is a polymer consisting of only one junction and arms diverging from the junction, as illustrated in Fig. 4.2.

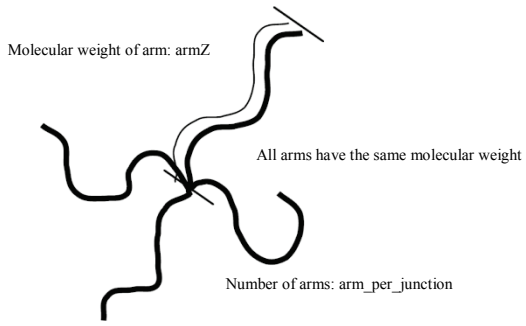

Fig. 4.2: Star polymer

For star polymers, it is necessary to give the number of arms diverging from the junction (parameter name: `arm_per_junction`) in addition to the molecular weight of the arms (parameter name: `armZ`). An example is given below.

Example for a star polymer

```
molecule_spec1=star
armZ1=5.0
arm_per_junction1=4
```

4.2.8 Asymmetric (Miktoarm) Star Polymers

Asymmetric star polymer is a polymer in which the lengths of the arms diverging from the junction are not uniform. Miktoarm star polymer is a star polymer with arms composed of different chemistries. For these polymers, the molecular weights, the number of arms, and the chemistries for the components must be given as follows.

Example for an asymmetric star polymer

```
molecule_spec1=astar
arm1Z1=5.0
arm2Z1=10.0
```

```
arm1_per_junction1=4
arm2_per_junction1=2
chemical_spec_for_arm1_of_component1=1
chemical_spec_for_arm2_of_component1=2
```

In this example, a molecule is described as illustrated in Fig. 4.3. For the interactions between different chemical species, refer to Section 4.2.1

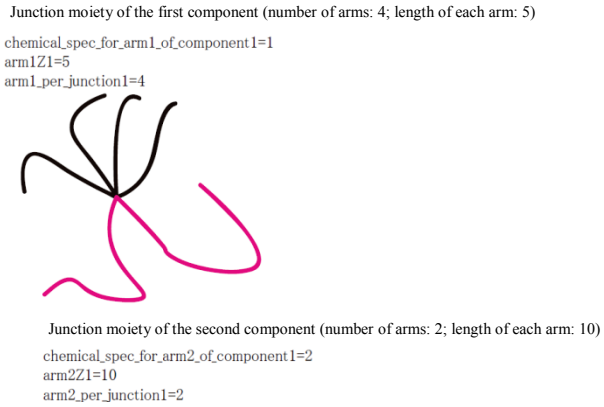

Fig. 4.3: Asymmetric (Miktoarm) star polymer

4.2.9 Pom-pom Polymers

In a pompom polymer, 2 star polymers are connected through a linear polymer as illustrated in Fig. 4.4.

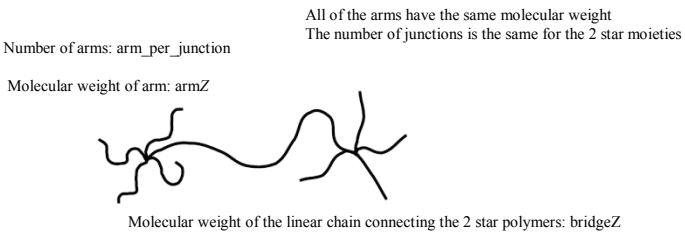

Fig. 4.4: Pompom polymer

The pompom polymer has been specified as the simplest class of multi-branched polymers by McLeich and Larson [6] in 1998. To describe the shape of this polymer, the length of the linear polymer (parameter name: `bridgeZ`) connecting the two star polymers is required, along with the description of the star polymers. An example of this description is as follows.

Example for a pompom polymer

```
molecule_spec1=pompom
armZ1=7.0
arm_per_junction1=3
bridgeZ1=15
```

4.2.10 Pom-pom Copolymers

A pompom polymer could be composed of two different chemistries for the star-part and the bridge-part, as described below. The chemical species specified by `chemical_spec_for_component` becomes the chemical species at the bridge moiety.

Example for a pompom copolymerized polymer

```
chemical_spec_for_component1=1
molecule_spec1=pompom
armZ1=7.0
arm_per_junction1=3
bridgeZ1=15
chemical_spec_for_arm_of_component1=2
```

4.2.11 Comb Polymers

In this class of branch polymers, multiple star polymers are connected through linear polymers as illustrated in Fig. 4.5. To specify the shape of this polymer, the number of junctions per molecule, `junction_number`, is required, in addition to the parameters for pompom polymers described in the previous sections. An example is shown below.

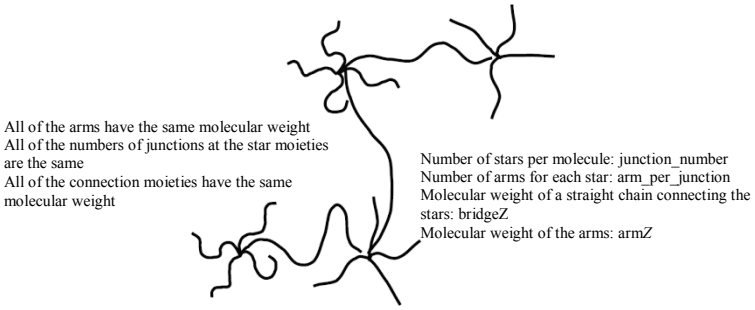

Fig. 4.5: Comb polymer

Example for a comb polymer

```
molecule_spec1=comb
armZ1=2
arm_per_junction1=3
bridgeZ1=10
junction_number1=3
```

4.2.12 Comb Copolymers

For a comb polymer with different chemistries for its arm-part and the bridge-part, the description is as shown below.

Example for a comb copolymerized polymer

```
molecule_spec1=comb
chemical_spec_for_component1=1
armZ1=5.0
arm_per_junction1=3
bridgeZ1=10
junction_number1=3
chemical_spec_for_arm_of_component1=2
```

4.2.13 Linear Copolymers

The description of linear polymers could be extended for diblock, or triblock copolymers. An example for a diblock copolymer is given as shown below.

Example for a diblock copolymer

```
molecule_spec1=diblock
Z1=20
block_ratio1_1=0.5
chemical_id_for_block1_1=1
chemical_id_for_block2_1=2
```

The parameter, `block_ratio`, should be set in between 0.0 and 1.0, with which the molecular weight of the block is given by multiplying the total molecular weight. Subsequently, the chemical composition is specified for each block by setting the parameters `chemical_id_for_block1_1` and `chemical_id_for_block2_1`

An example of a triblock copolymer is shown below. The polymer described here is illustrated in Fig. 4.6.

Example for a triblock copolymer

```
molecule_spec1=triblock
Z1=20
block_ratio1_1=0.3
block_ratio2_1=0.3
chemical_id_for_block1_1=1
chemical_id_for_block2_1=2
chemical_id_for_block3_1=1
```

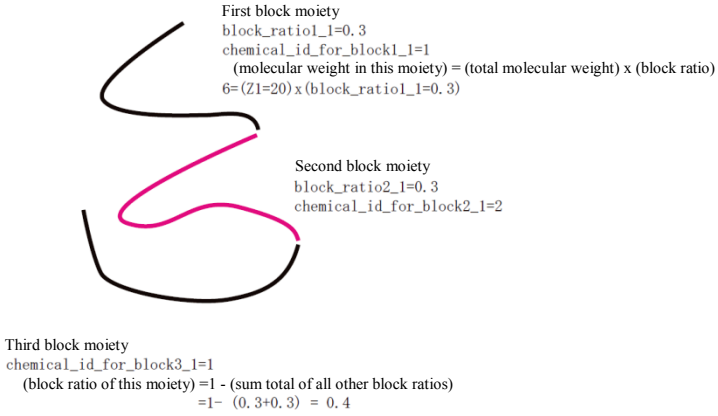

Fig. 4.6: Triblock copolymerized polymer

4.3 Determination of Z Values

In the simulations, the molecular weight is given by the Z value that is the number of entanglement segment. The Z number can be obtained for a specific molecular weight M and the entanglement molecular weight M_e, the latter depends on the chemistry of the polymer and its concentration. The value of M_e has been determined experimentally by the following formula [4].

$$M_e = \frac{\rho RT}{G_0} \tag{4.1}$$

Here, ρ is the density of the polymer, R the gas constant, T the temperature, and G_0 the plateau modulus. The NAPLES molecular weight Z can be determined by the following formula using the molecular weight M, and M_e . The prefactor 1.6 is related to the fluctuation around the entanglement [7, 8].

$$Z = \frac{M}{M_e} \times 1.6 \tag{4.2}$$

4.4 Setting Deformations and Flows

4.4.1 Fundamentals

In the settings for a flow or deformation, the type of flow or deformation is specified by the parameter `flow_type`, and parameters for each flow/deformation are specified as necessary. Step deformations and start-up flows can be selected for shear, uniaxial elongation, biaxial elongation, and planar elongation. The dynamic mode can be selected for shear deformation as well. The parameters will be defined below, and examples provided for each case. Mixed flows/deformations cannot be attained in the current version of the code.

4.4.2 Unit Time and Flow Rate

Because the characterictic time τ_e is used as the unit of time as mentioned in Section 6.5, the flow rate$1/\tau_e$ is the unit of flow rate. Accordingly, the value of τ_e is necessary to convert the actual flow rate to the simulation parameter. See Section 3.8.1 for the determination ofτ_e.

4.4.3 Equilibration

The initial structure of the system is not in the equilibrium state, and thus, the data is meaningless unless an adequate equilibration is attained. For this sake, it is recommended to add the following parameter shown below.

Setting equilibration time

```
pre_reptation_step=100
```

The flow or deformation is not applied to the system during the time given above. The recommended value is the longest relaxation time of the system. In the case of a linear polymer, the longest relaxation time τ_c is roughly expressed by $\tau_c = 0.0025Z^{3.6}\tau_e$. On the other hand, the calculation may be started without equilibration if the calculation is restarted from a given equilibrated configuration. In these cases, specify `pre_reptation_step=0`.

4.4.4 Step Deformations

A stepwise deformation can be applied by the following description in the input file.

Example for applying a step deformation

```
flow_type=step_shear
pre_reptation_step=100
initial_strain=0.5
quit_dynamics_count=10000
```

The flow field is set with the `flow_type` parameter. The following table shows the applicable flow types. Here, `pre_reptation_step` represents the time steps for equilibration, `initial_strain` the strain applied after the equilibration in terms of Hency strain, and `quit_dynamics_count` the time steps after the deformation.

Table 4.2: Specification of the flow type for a step deformation

Flow type	Input value
Shear	`step_shear`
Uniaxial	`uniaxal_step`
Biaxial	`biaxal_step`
Planar	`planar_step`

Note that the strain should be given in terms of Hency strain which is different from the engineering strain. For example, uniaxial strain ϵ is defined by the following formulation, where l_0 represents the initial length of the system and l the length after the elongation.

$$\epsilon = \exp \frac{l}{l_0} \tag{4.3}$$

4.4.5 Start-up Flows and Steady Flows

An example of the parameters for start-up flow calculation is given below.

Example for start-up flow

```
flow_type=start_up_shear
pre_reptation_step=100
strain_rate=0.05
quit_dynamics_count=10000
```

Here, the flow field is set with the `flow_type` parameter. The list of flow fields is shown in Table 4.3. `pre_reptation_step` represents the equilibration time steps, and `quit_dynamics_count` the time steps after the equilibration with the flow.

Table 4.3: Specification of flow type for start-up flow

Flow type	Input value
Shear	`start_up_shear`
Uniaxial	`uniaxal`
Biaxial	`biaxal`
Planar	`planar`

Note that the calculation of elongational flow cannot be restarted from the finalconf.npls file obtained by the calculation with `flow_type=equilibrium`. To obtain an equilibrated configuration for elongational flows, perform a calculation with no flow by setting `strain_rate=0` with the required flow field.

4.4.6 Dynamic Shear Deformations

Oscillatory shear deformations described as eq 4.4 can be applied by the parameters shown below.

$$\gamma = \gamma_0 \sin(\omega t) \tag{4.4}$$

Example for dynamic shear

```
flow_type=dynamic_shear
pre_reptation_step=100
strain_ampritude=0.25
strain_omega=0.1
dynamic_strain_cycle=5.0
```

Here, `pre_reptation_step` represents the time steps for equilibration, `strain_ampritude` and `strain_omega` represent γ_0 and ω in eq (4.4) respectively, and `dynamic_strain_cycle` represents the number of cycles of the oscillation. Note that the frequency is normalized with respect to the dimensionless time unit τ_e.

4.5 Setting Calculation Conditions

In this section, the parameters other than the sample settings and the flow settings are summarized.

4.5.1 Seed for Random Numbers

Due to the nature of stochastic simulations, the seed for pseudo-random number generator is of importance. Indeed, for the simulations with different initial configurations, the seed number must be different. The seed for random numbers is specified with the following parameter.

Parameter for setting the seed for random numbers

```
seed=[arbitrary integer value]
```

4.5.2 Unit Cell Size

NAPLES uses the periodic boundary conditions. The unit cell size can be specified by the following parameter. Here, the unit of length is the average length of the entangled segments.

Parameter for setting the size of the cell for the calculation

```
cell_size=[length of side of unit cell. 8 unless specified]
```

4.5.3 Output Files

The code outputs some files for the results of the calculation as given in Chapter 5. The control keys that can be specified as input parameters are as follows.

Parameters for controlling the output files

```
measure_flag=[interval for writing the stress file measure.npls. Default is set to 1
(recommended value)]
chain_av_flag=[interval for writing the statistics stat.npls file for the form of the
molecular chain. Default is set to 0 (Not written)]
snap_shot_save_flag=[interval for writing  the entire configuration for creating the
snapshot config[xxxx].npls file. Default is set to 0 (Not written)]
```

4.6 Parameter File Reference Manual

Table 4.4: Parameter reference

Parameter	Definition	Example
`calculation_mode`	Setting used at the start of the calculation. The key `new`, `restart` and `flow` initiate the calculations starting from scratch, from a given state in `finalconf.npls`, and from a given state in `finalconf.npls` with flow/deformations in the input file, respectively. The default setting is `new`.	`calculation_mode=new`
`cell_size`	Size of the unit cell for the periodic boundary condition. The default value is 8 (in units of the average segment length).	`cell_size=8`
`quit_dynamics_count`	Time steps to complete the calculation (in units of τ_e).	`quit_dynamics_count=10000`
`flow_type`	Parameter for specifying the flow field including `equilibrium` (without flow), `step_shear` (stepwise shear deformation), `uniaxal_step` (stepwise uniaxial elongation), `biaxal_step` (stepwise biaxial elongation), `planar_step` (stepwise planar elongation), `start_up_shear` (start-up shear flow), `uniaxal` (uniaxial elongational flow), `biaxal` (biaxial elongational flow), `planar` (planar elongational flow) and `dynamic_shear` (oscillatory shear deformation). The default is `equilibrium`.	`flow_type=uniaxal`
`pre_reptation_step`	Equilibration time without flow. The default is 100.	`pre_reptation_step=100`
`initial_strain`	Strain for stepwise deformations. The default is 0.	`Initial_strain=0.5`
`strain_rate`	Strain rate for flows, in units of $1/\tau_e$. The default is 0.	`strain_rate=0.01`
`strain_ampritude`	Strain amplitude for oscillatory shear. The default is 0.25.	`strain_ampritude=0.25`
`strain_omega`	Frequency for oscillatory shear, in units of $1/\tau_e$. The default is 0.	`strain_omega=0.01`
`dynamic_strain_cycle`	The number of strain cycles for the dynamic shear mode. The default is 5.	`dynamic_strain_cycle=2.0`
`chemical_component_number`	Number of chemical species contained in the sample. The default is 1.	`chemical_component_number=2`

Parameter	Definition	Example
chi12	Interaction parameter between chemical species.	chi12=2.0
bond_ratio_for_chemical[chemical id]	Segment length for the molecules with the chemistry identified by [chemical id]. The default is 1.0.	bond_ratio_for_chemical2=1.5
friction_ratio_for_chemical[chemical id]	Friction of entangled segment for the molecules with the identified by [chemical id]. The default is 1.0.	friction_ratio_for_chemical2=1.5
component_number	Number of constituents contained in the sample. The default is 1.	component_number=2
chemical_spec_for_component[component id]	Chemical species of component identified by [component id].	chemical_spec_for_component2=2
molecule_spec[component id]	Molecular topology of component identified by [component id] including linear, star, pompom, comb, astar diblock and triblock.	molecule_spec2=star
fraction[component id]	Volume fraction of component identified by [component id].	fraction1=1.0
Z[component id]	Molecular weight of linear chain component identified by [component id].	Z1=10
armZ[component id]	Molecular weight of arms for the component identified by[component id].	armZ1=5
arm1Z[component id]	Molecular weight of the first branched moiety of the asymmetric star polymer identified by[component id].	arm1Z1=3
arm2Z[component id]	Molecular weight of the second branched moiety of the asymmetric star polymer identified by [component id].	arm2Z1=5
bridgeZ[component id]	Molecular weight of a bridge between the branches of the pompom or comb polymer identified by [component id].	bridgeZ2=20
arm_per_junction[component id]	Number of arms per branch of the branched polymer identified by [component id].	arm_per_junction3=3
arm1_per_junction[component id]	Number of arms of the first branched moiety in the asymmetric star branched polymer identified by [component id].	arm1_per_junction3=3
arm2_per_junction[component id]	Number of arms of the second branched moiety in the asymmetric star branched polymer identified by [component id].	arm2_per_junction3=3
junction_number[component id]	Number of branches per molecule of the comb polymer identified by [component id].	junction_number4=3
chemical_spec_for_arm_of_component [component id]	Chemical composition of the branched moiety (arm) for the pompom or comb polymer identified by[component id].	chemical_spec_for_arm_of_component2=2
block_ratio1_[component id]	Block ratio of the first block moiety in the linear block copolymer identified by [component id].	block_ratio1_1=0.5
block_ratio2_[component id]	Block ratio of the second block moiety in the linear triblock copolymer identified by[component id].	block_ratio2_1=0.5
chemical_id_for_block1_[component id]	Chemical composition of the first block in the copolymer identified by[component id].	chemical_id_for_block1_1=1
chemical_id_for_block2_[component id]	Chemical composition of the second block in the copolymer identified by[component id].	chemical_id_for_block2_1=2

Parameter	Definition	Example
`chemical_id_for_block3_[component id]`	Chemical composition of the third block in the triblock copolymer identified by `[component id]`.	`chemical_id_for_block3_1=1`
`chain_av_flag`	Key for outputting the `stat.npls file`. The default is 0 (not output). When a numeral is specified, it represents the output time interval.	`chain_av_flag=10`
`measure_flag`	Key for outputting the `measure.npls` file. The default is 1. When a numeral is specified, it represents the output time interval.	`measure_flag=1`
`snapshot_save_flag`	Key for outputting the `config[time of saving].npls` file. The default is 0 (not output). When a numeral is specified, it represents the output time interval.	`snap_shot_save_flag=1000`
`FENE_flag`	Key for activating the FENE effect. The default is false, which attains Gaussian springs.	`FENE_flag=true`
`maximum_stretch_for_chemical [chemical id]`	The maximum elongation ratio when the FENE effect is specified. The default value is 4.	`maximum_stretch_for_chemical1=10`
`friction_reduction_flag`	Key for activating the SORF effect. The default is false, which attains simulations with constant frictions.	`friction_reduction_flag=true`

4.7 UDF File Reference

The parameters used in UDF format files are described in this section. Because the parameter names are almost identical to the NAPLES-specific parameters, for details refer to section 4.6.

Name	Type	Value	
input.udf		-	-
Chemical_Component[]	struct array	-	-
Chemical_Component[0]	struct	-	-
Name	string	chemical_A	
Id	int	1	
G0	double	2600000.0	[Pa]
G_nap	double	4160000.0	
Me_nap	double	647.15	[g/mol]
tau_d	double	0.0	[s]
chi12	double	0.0	
bond_ratio_for_chemical	double	0.0	
friction_ratio_for_chemical	double	0.0	
Component[]	struct array	-	-
Component[0]	struct	-	-
Simulation_Conditions	struct	-	-
Simulation_Parameters	struct	-	-

Fig. 4.7: Example screen when UDF files are opened with Gourmet

Chemical_Component[]

Chemical species are set as an array in UDF input files. The number of chemical species is automatically recognized as the array size and therefore does not need to be set explicitly.

- Name: Name of the arbitrary chemical species.
- Id: Specify 1 or 2.
- G0: Plateau modulus G0 [Pa]. Not used.
- G_nap: Unit of stress [Pa] in NAPLES. Optional.

- Me_nap: Entanglement molecular weight Me [g/mol]. Not used.
- tau_d: Longest relaxation time [s]. Not used
- tau_e: Unit time [s]. Optional.
- chi12: Interaction parameter between chemical species 1 and 2.
- bond_ratio_for_chemical: Ratio of the entangled segment length of chemical species 1 to 2.
- friction_ratio_for_chemical: Ratio of the coefficient of friction per segment of chemical species 1 to 2.
- FENE_flag: Key for controlling the extended effect. The default is false, and the effect is not introduced by default.
- maximum_stretch_for_chemical[]: Specifies the maximum elongation ratio when the extended effect is specified. The default value is 4.
- friction_reduction_flag: Key for controlling molecular friction depending on the elongational orientation of the molecule. The default is false, and friction is not introduced by default.

Component[]

The constituents are set as an array in UDF input files. The number of constituents is automatically recognized as the array size, and therefore does not need to be given explicitly.

- fraction: Volume fraction.
- molecule_spec: Molecular shape (linear, star, pompom, comb, astar, diblock, triblock).
- linear.chemical_spec_for_component: Chemical species number for the component.
- linear.Z: Molecular weight of a linear chain component.
- star.chemical_spec_for_component: Chemical species number for the component.
- star.armZ: Molecular weight of the arm.
- star.arm_per_junction: Number of arms from the branch point.
- astar.chemical_spec_for_arm1_of_component: Identification number of chemistry of the 1st branched moiety.
- astar.arm1Z: Molecular weight of the 1st branched moiety.
- astar.arm1_per_junction: Number of arms for the 1st branched moiety
- astar.chemical_spec_for_arm2_of_component: Identification number of chemistry of the 2nd branched moiety.
- astar.arm2Z: Molecular weight of the 2nd branched moiety.
- astar.arm2_per_junction: Number of arms for the 2nd branched moiety.
- pompom.chemical_spec_for_component: Identification number of chemistry of the main chain.
- pompom.armZ: Molecular weight of the arm .
- pompom.arm_per_junction: Number of arms from the branch point.
- pompom.chemical_spec_for_arm_of_component: Identification number of chemistry for the arm moiety.
- pompom.bridgeZ: Molecular weight of the bridge moiety.
- comb.chemical_spec_for_component: Identification number of chemistry of the component.
- comb.armZ: Molecular weight of the arm.
- comb.arm_per_junction: Number of arms from the branch point.
- comb.chemical_spec_for_arm_of_component: Identification number of chemistry for the arm moiety.
- comb.bridgeZ: Molecular weight of the bridge moiety.

- comb.junction_number: Number of branches per molecule.
- diblock.chemical_id_for_block1_: Identification number of chemistry for the 1st block moiety.
- diblock.block1Z: Molecular weight of the 1st block moiety.
- diblock.chemical_id_for_block2_: Identification number of chemistry for the 2nd block moiety.
- diblock.block2Z: Molecular weight of the 2nd block moiety.
- diblock.block_ratio1_: Block ratio of the 1st block moiety.
- tri_block.chemical_id_for_block1_: Identification number of chemistry for the 1st block moiety.
- tri_block.block1Z: Molecular weight of the 1st block moiety.
- tri_block.chemical_id_for_block2_: Identification number of chemistry for the 2nd block moiety.
- tri_block.block2Z: Molecular weight of the 2nd block moiety.
- tri block.chemical_id_for_block3_: Identification number of chemistry for the 3rd block moiety.
- tri_block.block3Z: Molecular weight of the 3rd block moiety.
- tri_block.block_ratio1_: Block ratio of the 1st block moiety.
- tri_block.block_ratio2_: Block ratio of the 2nd block moiety.

Simulation_Conditions

- seed: Starting seed value for random numbers.
- cell_size: Cell size.
- quit_dynamics_count: Total calculation time.
- pre_reptation_step: Equilibration time without a flow applied.
- flow_type: Selection of the flow field.
- step_shear.initial_strain: Initial strain for step shear deformations.
- uniaxal_step.initial_strain: Initial strain for step uniaxial stretches.
- biaxal_step.initial_strain: Initial strain for step biaxial stretches.
- planar_step_step.initial_strain: Initial strain for step planner stretches.
- start_up_shear.strain_rate: Strain rate for start-up shear flows.
- uniaxal.strain_rate: Strain rate for start-up uniaxial deformations.
- biaxal.strain_rate: Strain rate for start-up biaxial deformations.
- planar.strain_rate: Strain rate for start-up planner deformations.
- dynamic_shear.strain_ampritude: Strain amplitude in dynamic shear deformations.
- dynamic_shear.strain_omega: Frequency in dynamic shear deformations.
- dynamic_shear.dynamic_strain_cycle: Number of strain cycles in in dynamic shear deformations.
- chain_av_flag: Key for outputting the stat.npls file.
- measure_flag: Key for outputting the measure.npls file.
- snapshot_save_flag: Key for outputting the config[time of saving].npls file.

Simulation_Parameters

- restart_flag: Restart flag (0: new, 1: restart, 2: deform).

- simulation_type: Calculation mode for normal or frequency_scan.
- frequency_scan.wmin: Minimum frequency for frequency scanning calculations.
- frequency_scan.wmax: Maximum frequency for frequency scanning calculations.
- frequency_scan.div: Number of divisions in one decade for frequency scanning calculations.

Output

- movelog_out: Status report file.
- prmcheck_out: Parameter check file.
- mesure_out: Stress output file.
- stat_out: Molecular statistics output file.
- finalconf_out: Final configuration save file.
- config_out: Configuration save file.
- bistress_out: Save file for time development of biaxial elongational stress.
- bistress_smooth_out: Smoothed result of bistress_out
- plstress_out: Save file for time development of planar elongational stress.
- plstress_smooth_out: Smoothed result of plstress_smooth_out
- shearstress_out: Save file for time development of shear stress.
- shearstress_smooth_out: Smoothed result of shearstress_out
- unistress_out: Save file for time development of uniaxial elongational stress.
- unistress_smooth_out: Smoothed result of unistress_out
- gt_out: Save file for the linear shear relaxation modulus.
- gt_smooth_out: Smoothed result of gt_out
- et_out: Save file for the linear shear viscosity growth curve.
- et_smooth_out: Smoothed result of et_out
- gw_out: Save file for the dynamic viscoelasticity results obtained from gt_out.
- gw_smooth_out: Smoothed result of gw_out
- gwdirect_out: Save file for the dynamic viscoelasticity obtained vi naplesgw

Chapter 5 NAPLES Output Files

Various files are created by NAPLES and by the analysis tools. If the accessory analysis tools are used as-is according to Section 2.4, the respective result files are defined as follows. These files are automatically overwritten when the next analysis process is performed, and therefore these files should be saved as necessary.

When UDF format input files are used, the respective output files are linked from the UDF input files. Depending on the actions placed on the UDF input files, it is possible to call stress-analysis and plotting functions.

- bistress.npls, bistress_smooth.npls: Files containing the time development of biaxial elongational stress, and the smoothed results. For details about the contents, refer to Section 5.7.

- configXXXX.npls: File for saving the configuration. This file can be used for restarting the calculation. For details, refer to Section 5.6.
- et.npls, et_smooth.npls: Files containing the linear shear viscosity growth curve, and the smoothed results. For details, refer to Section 5.9.
- finalconf.npls: File for saving the final configuration for restarting the calculation. For details, refer to Section 5.5.
- gt.npls, gt_smooth.npls: Files containing the linear relaxation modulus, and the smoothed results. For details, refer to Section 5.8.
- gw.npls, gw_smooth.npls: Files containing the dynamic viscoelasticity (storage modulus G' and loss modulus G") and the smoothed results. For details, refer to Section 5.10.
- gwdirect.npls: Dynamic viscoelasticity curve data (storage modulus G' and loss modulus G"). For details about the contents, refer to Section 5.10.
- input.npls: Default input parameter file used when the Input file is not specified as an argument at startup. For details, refer to Section 4.
- measure.npls: Output file for recording the stress during the simulations. For details, refer to Section 5.3.
- movelog.npls: Log file for the calculation. For details about the contents, refer to Section 5.1.
- plstress.npls, plstress_smooth.npls: File containing the time development of planner elongational stress, and the smoothed results.
- prmcheck.npls: Parameter checking file. For details, refer to Section 5.2.
- shearstress.npls, shearstress_smooth.npls: File containing the time development of shear stress, and the smoothed results.
- stat.npls: File containing the molecular statistics. For details, refer to Section 5.4.
- tmp.npls: Temporary working file created during the analysis. This file may be deleted.
- unistress.npls, unistress_smooth.npls: File containing the time development of uniaxial elongational stress, and the smoothed results.

5.1 Status Report File movelog.npls

The status report file, `movelog.npls`, is created in the same directory as naplesOCTA(.exe) shortly after the code is executed. The contents are as follows.

Contents of movelog.npls at the execution

```
-------------------------
NAPLES is started with parameters in input.npls Thu Oct 17 18:32:40 2002
Initialization started. Thu Oct 17 18:32:40 2002
Main calculation started. 0 Thu Oct 17 18:32:46 2002
EQUILIBRIUM_CALC Thu Oct 17 18:32:46 2002
```

The `movelog.npls` file records the date and time when the event occured, together with error messages and some logs. For instance, when a flow deformation is applied, the process is recorded as shown below.

Contents of `movelog.npls` at a step shear deformation

```
SINGLE_STEP_CALC Thu Oct 17 18:44:37 2002
PRE_RUN 0 Thu Oct 17 18:44:37 2002
STEP_SHEAR_APPLIED 3 Thu Oct 17 18:44:59 2002
END 10 Thu Oct 17 18:46:08 2002
```

When there is an error in the contents of the input parameter file, a message is generated as shown below.

Message in `movelog.npls` when there is an error in the Input parameter file

```
Description in the data file, [xxx], is meaningless Thu Oct 17 18:37:36 2002
```

This warns the user that the parameter `xxx` in the input parameter file is either invalid, or has an invalid value. If the input parameter file is not specified and the input.npls file cannot be read, the following message is output, and NAPLES starts with the conditions pre-embedded in the program.

Message in `movelog.npls` showing that NAPLES is started with the parameters embedded in the code

```
Parameters are set as default Thu Oct 17 18:39:36 2002
```

The start-up parameters in this case could be seen in `prmcheck.npls` file as described in the next section.

5.2 Parameter Check File prmcheck.npls

`prmcheck.npls` file is automatically created at the starting of the code for damping the parameters. For details about the parameters, refer to Section 4.7.

5.3 Stress Output File measure.npls

The stress output file, measure.npls, records the time development of the deformation and the stress, as shown below.

Example of `measure.npls`

```
0     0.00e+00     -3.060009e-02   1.820663e+00   1.652494e+00   2.459011e+00
1     0.00e+00     -1.483576e-02   1.167096e+00   1.146231e+00   1.269184e+00
2     0.00e+00     -7.436174e-03   1.107850e+00   1.144368e+00   1.191704e+00
3     0.00e+00     -1.660099e-02   1.163289e+00   1.137091e+00   1.193193e+00
0     0.00e+00     5.519445e-01    1.430961e+00   1.137091e+00   1.193193e+00
1     0.00e+00     1.126641e-01    1.273881e+00   1.186699e+00   1.205338e+00
2     0.00e+00     8.506818e-02    1.227362e+00   1.163581e+00   1.177303e+00
3     0.00e+00     4.590036e-02    1.193680e+00   1.152211e+00   1.175619e+00
4     0.00e+00     3.322798e-02    1.158326e+00   1.172665e+00   1.141329e+00
```

The 1st row represents the calculation time on a scale of τ_e, the unit time used in the code. The 2nd row represents the total strain applied to the system. The 3rd row represents σ_{12} (shear stress), and the 4th to 6th rows represent σ_{11}, σ_{22}, and σ_{33} (normal stress). These stress values are scaled to the values in NAPLES and are dimensionless quantities.

When the calculated system has multiple constituents, the stress data is calculated for each constituent, and the rows are added to the file for each one. For example, when a system consisting of 2 components is calculated, the stress of the entire system is written in the 3rd to 6th rows, the stress generated by the 1st component is written in the 7th to 10th rows, and the stress generated by the 2nd component is written in the 11th to 14th rows.

5.4 Chain Statistics File stat.npls

The time development of the number of entanglements per chain, the average distance between entanglement points, and the end-to-end distance are stored in `stat.npls`.

Example of `stat.npls`

```
  0   4.7709  1.0272 0.759200 0.226836       3.380538       8.335377
 10   4.9818  1.1555 0.706237 0.187129       3.094384       6.412024
```

The 1st row represents the calculation time on a scale of τ_e. The 2nd row represents the average number of segments per chain; the 3rd row the variance; the 4th and 5th rows represent the average segment length, and its variance; respectively, and the 6th and 7th rows represent the average squared end-to-end distance and its variance, respectively.

When the calculated system has multiple constituents, the data is calculated for each constituent, excluding the calculation time in the 1st row, and the rows are added to the file for every constituent.

5.5 Configuration Save Files

There are configuration save files in which the simulation snapshot is stored for restarting calculations. The file named `finalconf.npls` is automatically generated and overwritten in every 100 steps. The files with the name `config[xxx].npls` are the snapshot files at the certain time step xxx.

5.6 Stress Data Files

By the analysis tools described in Section 3, obtained measure.npls file can be converted to some rheological measures, and the obtained measures are recorded in the files below. .

- bistress.npls, bistress_smooth.npls: Time development of biaxial elongational stress, and the smoothed results
- plstress.npls, plstress_smooth.npls: Time development of planar elongational stress, and the smoothed results
- shearstress.npls, shearstress_smooth.npls: Time development of shear stress, and the smoothed results
- unistress.npls, unistress_smooth.npls: Time development of uniaxial elongational stress, and the smoothed results

The content is as follows.

Example of bistress.npls

```
0.000000e+00 3.933300e-02
1.000000e+00 5.783300e-02
2.000000e+00 9.034000e-03
3.000000e+00 2.207400e-02
4.000000e+00 4.046700e-02
5.000000e+00 3.910700e-02
```

Here, the first row represents time, and the second row represents stress. The units of time and stress are dimensionless in the NAPLES calculation system. To give units to the time and stress, refer to Section 6.5. To convert the stress to viscosity or elastic modulus, use the following relation in the same manner as in experiments.

$$\eta = \sigma/\dot{\epsilon} \tag{5.1}$$

$$G = \sigma/\epsilon \tag{5.2}$$

In this formula, η represents viscosity, σ represents stress, G represents elastic modulus, ϵ represents strain, and $\dot{\epsilon}$ represents the strain rate.

5.7 Linear Relaxation Modulus Files gt.npls and gt_smooth.npls

The linear relaxation modulus is calculated from the stress fluctuation as explained in Section 3.2. The contents of the obtained files are as shown below. The first row represents time, and the second row represents the change in relaxation modulus over time. It must be noted that the second row represents elastic modulus, not stress.

Example of gt.npls

```
0 1.000000e+00
1 4.879472e-01
2 4.173991e-01
3 3.699413e-01
4 3.302976e-01
5 3.034629e-01
6 2.775755e-01
```

5.8 Linear Shear Viscosity Growth Files et.npls and et_smooth.npls

The linear viscosity growth curve can be obtained from linear relaxation modulus as explained in Section 3.5. The obtained file is as shown below. The first row represents time, and the second row represents the linear shear viscosity. It must be noted that the second row represents viscosity, not stress. The linear viscosity curves under elongational deformations can also be obtained according to the Trouton's rule.

Example of et.npls

```
1 0.00140398
2 0.0141274
3 0.0120504
4 0.00352317
5 0.00572446
```

5.9 Dynamic Viscoelasticity Files gw.npls, gw_smooth.npls, and gwdirect.npls

As explained in Section 3.3, the dynamic viscoelastic modulus, G' and G", can be obtained from the linear relaxation modulus or from the stress obtained under oscillatory deformations. If they are obtained from the linear relaxation modulus, the result file is `gw.npls` and the contents are as shown below. The first row represents frequency, the second row represents the storage modulus G', and the third row represents the loss modulus G". In the case of simulations under oscillatory field, the output file is `gwdirect.npls` that is formatted in a similar manner.

Contents of gw.npls (Example)

```
8.000000e-01  1.052871e-01  8.982074e-02
6.000000e-01  1.059345e-01  8.313399e-02
4.000000e-01  9.529423e-02  7.436524e-02
3.000000e-01  8.005790e-02  6.357562e-02
2.000000e-01  6.599811e-02  6.273698e-02
1.000000e-01  4.031614e-02  5.399160e-02
8.000000e-02  3.233802e-02  4.980498e-02
```

Chapter 6 Theoretical Background

It is practically difficult in existing molecular simulators (including the bead-spring type simulations [9]) to explore entangled polymer dynamics and rheology for the following reasons: 1) the molecular weights range over several thousands to millions; and 2) slow dynamics with a characteristic time of several hundred seconds are of significance. Due to these reasons, molecular models with higher level of coarse-graining than the bead-spring description have been being developed. The most successive approach is the tube models [10] in which the entangled polymer dynamics is replaced by the single chain motion confined in a tube. Although the single-chain description attains computational efficiencies, there are fundamental difficulties for implementation of multi-chain effects. NAPLES is a simulator that incorporates various multi-chain effects in the same manner as in existing molecular simulators, while conducting coarse-graining similar to the reptation theory. In this chapter, its theoretical background is briefly given.

NAPLES is a code that numerically solve the kinetic equations constructed for the primitive chain network model [11]. In the model, entangled polymers are cast into networks that consist of node, strand, and dangling ends. Each polymer chain corresponds to the paths in the network connecting two dangling ends through several strands and nodes. At each node, there exists a slip-link that bundles two polymer chains in pair. The slip-link allows sliding of the polymers whereas it restricts the motion in perpendicular directions to the polymer backbone. When a polymer slides off from the slip-link at the chain end, the slip-link is removed from the system to release the coupled polymer. On the other hand, when a polymer protrudes from the slip-link at the end with a certain amount, another polymer randomly chosen from the surroundings is hooked and bundled by the newly created slip-link. The state valuables are the position of slip-links $\{\mathbf{R}\}$, the number of Kuhn-steps assigned on the strands $\{n\}$, and the number of slip-links on the polymers $\{Z\}$. Kinetics of these valuables are described below.

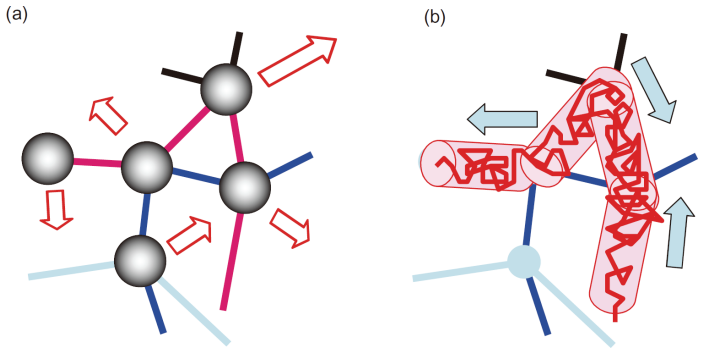

Fig. 6.1: Schematic of the model. (a) Motion of entanglement points. (b) Sliding of a polymer chain.

The dynamics of {R} shown in Fig 6.1 (a) is described by the Langevin-type equation of motion shown below.

$$\zeta \frac{d\boldsymbol{R}_i}{dt} = \frac{3kT}{b^2} \sum_j^4 \frac{\boldsymbol{r}_j}{n_j} - n_0 \nabla \mu + \boldsymbol{F} \tag{6.1}$$

Here, the LHS represents the drag force and ζ is the friction coefficient of the node (slip-link). The first term on RHS is the balance of tension acting on the strands diverging from the node. kT is the thermal energy, b the Kuhn length, $\mathbf{r}$ the bond vector of strands, n the number of Kuhn segments on each strand. The 2nd term on RHS represents the osmotic force due to the density fluctuation in the system. Here, n0 is the average number of n and μ represents the chemical potential. The latter is given by

$$\mu(\boldsymbol{R}) = \frac{\partial F}{\partial n(\boldsymbol{R})} \tag{6.2}$$

The free energy F is defined in a phenomenological manner to suppress the density fluctuation. The factor ϵ is empirically determined at 0.5.

$$\frac{F}{kT} = \begin{cases} \epsilon \left(\frac{n(\boldsymbol{R})}{\langle n \rangle} - 1 \right)^2 & \text{for} \quad n(\boldsymbol{R}) > \langle n \rangle \\ 0 & \text{for} \quad n(\boldsymbol{R}) \leq \langle n \rangle \end{cases} \tag{6.3}$$

The 3rd term on RHS of equation 6.1 is a random force obeying

$$\langle \boldsymbol{F} \rangle = \boldsymbol{0} \tag{6.4}$$

$$\langle \boldsymbol{F}_i(t) \cdot \boldsymbol{F}_j(t') \rangle = 6kT\zeta\delta_{ij}\delta(t-t') \tag{6.5}$$

The kinetics of $\{n\}$ is described by the rate equation given below.

$$\frac{\zeta}{2}\frac{1}{\rho}\frac{dn_i}{dt} = \frac{3kT}{b^2}\left(\frac{r_{i+1}}{n_{i+1}} - \frac{r_i}{n_i}\right) - n_0 \mathrm{grad}\mu + f \tag{6.6}$$

$$\rho = \frac{1}{2}\left(\frac{n_{i+1}}{r_{i+1}} + \frac{n_i}{r_i}\right) \tag{6.7}$$

The LHS is the drag force and ρ represents the linear monomer density as defined in equation 6.7. The RHS consists of the balance of tension, the osmotic force and the random force. Because the equation is defined in 1-D, the random force here is defined in 1-D to obey the following statistics.

$$\langle f_i \rangle = 0 \tag{6.8}$$

$$\langle \boldsymbol{f}_i(t)\boldsymbol{f}_j(t') \rangle = kT\zeta\delta_{ij}\delta(t-t') \tag{6.9}$$

The kinetics of $\{Z\}$ is attained via the algorism for creation/destruction of slip-links around chain ends. Following the spirit of the tube framework, the chain is disentangled only when the chain end passes through the slip-link. Vice versa, when the chain end protrudes beyond a certain amount, the chain end is entangled with another chain. These topological changes are triggered by the number of monomers at chain ends, n. When n exceeds the range shown in equation 6.10, the creation/ destruction of the entanglement is performed.

$$\frac{1}{2} < \frac{n}{n_0} < 1 + \frac{1}{2} \tag{6.10}$$

In the simulations, the equations shown above are numerically integrated with non-dimensional form in which the unit of length, energy and time are taken as $a = b\sqrt{n_0}$, kT and $\tau_e = \zeta a^2/kT$. Figure 6.2 shows a typical snapshot of the simulation based on the primitive chain network model.

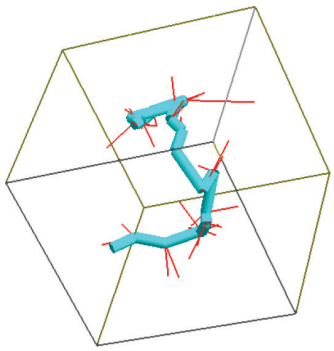

Fig. 6.2: Typical snap shot of the primitive chain network simulation.

NAPLES simulations have been made for various problems in entangled polymer systems. For branched polymers, there are publications for star polymers[12, 13], comb branched polymers[14] and pom-pom branched polymers[15]. The simulations have been made for the systems with molecular weight distributions as well[16–18]. Under shear deformations, the simulations under fast

flows[12, 19, 20] and under large step shear deformations [21–24] have been made. Meanwhile, for elongational flows, a series of studies have been attempted for the change of friction[5, 17, 25–28].

References

1. Masubuchi Y (2014) Simulating the Flow of Entangled Polymers. Annu Rev Chem Biomol Eng 5:11–33. doi: 10.1146/annurev-chembioeng-060713-040401
2. Masubuchi Y (2016) Molecular Modeling for Polymer Rheology. In: Ref. Modul. Mater. Sci. Mater. Eng. Elsevier, pp 1–7
3. JACI (2016) Computer Simulation of Polymeric Materials. doi: 10.1007/978-981-10-0815-3
4. Fetters LJ, Lohse DJ, Richter D, et al (1994) Connection between polymer molecular weight, density, chain dimensions, and melt viscoelastic properties. Macromolecules 27:4639–4647. doi: 10.1021/ma00095a001
5. Yaoita T, Isaki T, Masubuchi Y, et al (2012) Primitive chain network simulation of elongational flows of entangled linear chains: Stretch/orientation-induced reduction of monomeric friction. Macromolecules 45:2773–2782. doi: 10.1021/ma202525v
6. McLeish TCB, Larson RG (1998) Molecular constitutive equations for a class of branched polymers: The pom-pom polymer. J Rheol (N Y N Y) 42:81. doi: 10.1122/1.550933
7. Masubuchi Y, Ianniruberto G, Greco F, Marrucci G (2003) Entanglement molecular weight and frequency response of sliplink networks. J Chem Phys 119:6925–6930. doi: 10.1063/1.1605382
8. Masubuchi Y, Ianniruberto G, Greco F, Marrucci G (2008) Quantitative comparison of primitive chain network simulations with literature data of linear viscoelasticity for polymer melts. J Nonnewton Fluid Mech 149:87–92.
9. Kremer K, Grest GS (1990) Dynamics of entangled linear polymer melts: A molecular-dynamics simulation. J Chem Phys 92:5057. doi: 10.1063/1.458541
10. Doi M, Edwards S (1988) The theory of polymer dynamics.
11. Masubuchi Y, Takimoto J-II, Koyama K, et al (2001) Brownian simulations of a network of reptating primitive chains. J Chem Phys 115:4387. doi: 10.1063/1.1389858
12. Masubuchi Y, Ianniruberto G, Greco F, Marrucci G (2004) Molecular simulations of the long-time behaviour of entangled polymeric liquids by the primitive chain network model. Model Simul Mater Sci Eng 12:S91.
13. Masubuchi Y, Yaoita T, Matsumiya… Y, et al (2011) Primitive chain network simulations for asymmetric star polymers. J Chem … 134:194905. doi: 10.1063/1.3590276
14. Masubuchi Y, Matsumiya Y, Watanabe H, et al (2012) Primitive chain network simulations for comb-branched polymer under step shear deformations. Rheol Acta 51:1–8. doi: 10.1007/s00397-011-0574-x
15. Masubuchi Y, Matsumiya Y, Watanabe H, et al (2014) Primitive chain network simulations for Pom-Pom polymers in uniaxial elongational flows. Macromolecules 47:3511–3519. doi: 10.1021/ma500357g
16. Masubuchi Y, Watanabe H, Ianniruberto G, et al (2008) Comparison among Slip-Link Simulations of Bidisperse Linear Polymer Melts. Macromolecules 41:8275–8280. doi: 10.1021/ma800954q
17. Takeda K, Sukumaran SK, Sugimoto M, et al (2015) Primitive chain network simulations for elongational viscosity of bidisperse polystyrene melts. Adv Model Simul Eng Sci 2:11. doi: 10.1186/s40323-015-0035-7
18. Masubuchi Y, Amamoto Y (2016) Orientational Cross-Correlation in Entangled Binary Blends in Primitive Chain Network Simulations. Macromolecules 49:9258–9265. doi: 10.1021/acs.macromol.6b01642
19. Masubuchi Y, Furuichi K, Horio K, et al (2009) Primitive chain network simulations for entangled DNA solutions. J Chem Phys 131:114906. doi: 10.1063/1.3225994
20. Masubuchi Y, Watanabe H (2014) Origin of stress overshoot under start-up shear in primitive chain network simulation. ACS Macro Lett 3:1183–1186. doi: 10.1021/mz500627r
21. Furuichi K, Nonomura C, Masubuchi Y, et al (2008) Entangled polymer orientation and stretch under large step shear deformations in primitive chain network simulations. Rheol Acta 47:591–599. doi: 10.1007/s00397-008-0258-3
22. Furuichi K, Nonomura C, Masubuchi Y, et al (2007) Primitive chain network simulations of damping functions for shear, uniaxial, biaxial and planar deformations. 日本レオロジー学会誌 35:73–77.
23. Furuichi K, Nonomura C, Masubuchi Y, Watanabe H (2010) Chain contraction and nonlinear stress damping in primitive chain network simulations. J Chem Phys 133:174902. doi: 10.1063/1.3502681
24. Furuichi K, Nonomura C, Masubuchi Y, Watanabe H (2013) Nonlinear Stress Relaxation of Scarcely Entangled Chains in Primitive Chain Network Simulations. Nihon Reoroji Gakkaishi 41:13–19. doi: 10.1678/rheology.41.13
25. Yaoita T, Isaki T, Masubuchi Y, et al (2011) Primitive chain network simulation of elongational flows of entangled linear chains: Role of finite chain extensibility. Macromolecules 44:9675–9682. doi: 10.1021/ma202166y
26. Takeda K, Sukumaran SK, Sugimoto M, et al (2015) Test of the Stretch/Orientation-Induced Reduction of Friction for Biaxial Elongational Flow via Primitive Chain Network Simulation. Nihon Reoroji Gakkaishi 43:63–39. doi: 10.1678/rheology.43.63

27. Masubuchi Y, Matsumiya Y, Watanabe H (2014) Test of Orientation / Stretch-Induced Reduction of Friction via Primitive Chain Network Simulations for Polystyrene , Polyisoprene , and Poly (n ‐ butyl acrylate). Macromolecules 47:6768–6775. doi: 10.1021/ma5016165
28. Bhattacharjee PK, Nguyen DA, Masubuchi Y, Sridhar T (2017) Extensional Step Strain Rate Experiments on an Entangled Polymer Solution. Macromolecules. doi: 10.1021/acs.macromol.6b00823