OCTA

Integrated simulation system for soft materials

RHEOLOGY SIMULATOR

PASTA

USER'S MANUAL

OCTA User's Group

DEC. 25 2002

Authors of the Manual

Chapter 1	Jun-ichi Takimoto, Hiroyasu Tasaki
Chapter 2	Jun-ichi Takimoto, Hiroyasu Tasaki
Chapter 3	Jun-ichi Takimoto, Hiroyasu Tasaki
Chapter 4	Jun-ichi Takimoto, Hiroyasu Tasaki
Chapter 5	Jun-ichi Takimoto, Hiroyasu Tasaki
Chapter 6	Jun-ichi Takimoto, Tatsuya Shoji
Chapter 7	Jun-ichi Takimoto, Hiroyasu Tasaki
Appendix A	Jun-ichi Takimoto, Hiroyasu Tasaki

Programers

PASTA	Jun-ichi Takimoto
FORK	Tatsuya Shoji

Acknowledgment

This work is supported by the national project, which has been entrusted to the Japan Chemical Innovation Institute (JCII) by the New Energy and Industrial Technology Development Organization (NEDO) under METI's Program for the Scientific Technology Development for Industries that Creates New Industries.

Contents

List of Figures

Chapter 1

What is PASTA ?

Recently, it is becoming possible to control the molecular weight distribution and the branching structure of polymeric materials not only in laboratories but also in the industry. Processability of polymeric materials is strongly influenced by their rheological properties, which in turn are governed by the molecular weight distribution and the branching structure. In order to aid the design of polymers with good processability, we have been developing a new stochastic simulation method that can directly calculate the rheological properties from the knowledge of the molecular weight distribution and the branching structure.

The program we have developed is called **PASTA** (*Polymer rheology Analyzer with Slip-link model of enTAnglement*). Current version of **PASTA** is able to calculate the rheological properties of monodisperse and polydisperse linear and star polymers and their mixtures.

FORK is a program for generating various molecular weight distributions. It creates input files of **PASTA** using the generated molecular weight distribution, along with the simulation conditions specified by the user.

Chapter 2

Theoretical background

2.1 Simulation model

The simulation method is based on the tube model which has become the standard model of polymer rheology[1]. The tube model, in its simplest form, makes two basic assumptions: the length of the primitive path is constant, and obstacles due to entanglements have an infinite life time. Although this "classical" tube model provides clear physical picture of the dynamics of entangled polymers, it does not work well in some cases. For example, the theory predicts that the shear stress under fast flow is a decreasing function of the shear rate. In order to overcome these difficulties, two important extensions have been proposed. One is the contour length fluctuation (CLF) and the other is the constraint release (CR). However, it would be very complicated to integrate all the effects into an analytical theory. Therefore, we use a stochastic simulation method which takes account of the reptation, CLF, and CR in a very simple way[2].

Each polymer chain is modeled by a primitive path and slip links along it, as shown in Fig.2.1. Suppose there are n slip links and let $\mathbf{r}_i (i = 1, 2, ..., n-1)$ be the subchain vector connecting the neighboring slip links. Also let s_1 and s_2 be the lengths of the tails at the ends of the primitive path. Then the total length of the tube L_t and the primitive path L are given by $L_t = \sum_{i=1}^{n-1} |\mathbf{r}_i|$ and $L = L_t + s_1 + s_2$. A collection of these chains is prepared in a computer, but their mutual position in the 3-dimensional space is not taken into account. Instead, each slip link in the system is paired with its partner which is randomly selected from the slip links on other chains. Fig.2.2 shows a schematic diagram of "pairing" of the slip links. The slip links constrain a pair of chains, and disappear when either one of the chain moves away from the slip link.

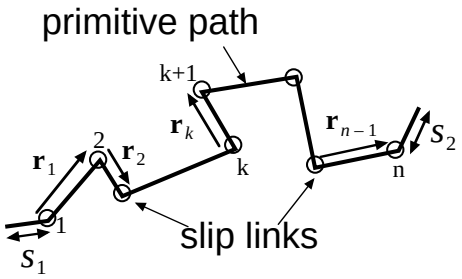

Figure 2.1: Schematic diagram of a primitive path and slip links in the simulation model.

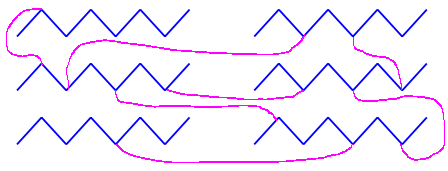

Figure 2.2: Pairing of the slip links. All the slip links have their partners, but only representative pairs are shown.

In each time step, the following four operations are performed.

1. Affine deformation of the tubes
   Each primitive path is deformed affinely by the macroscopic deformation of the sample.

2. Contour length fluctuation(CLF)

The length of each primitive path is updated according to the following Langevin equation:

$$\frac{dL}{dt} = -\frac{1}{\tau_R}(L(t) - L_{eq}) + \dot{L}_{affine} + g(t), \tag{2.1}$$

where $L_{eq} = Za$ is the equilibrium length of the primitive path, $Z = M/M_e$ is the average number of slip links, M is molecular weight of this chain, M_e is the entanglement molecular weight of this polymer, a is the average spacing between neighboring slip links which corresponds to the average diameter of the tube, $\tau_R = \tau_e Z^2$ is the Rouse relaxation time of this chain, and τ_e is the unit of time. $\dot{L}_{affine}$ is the time derivative of the tube length due only to the affine deformation, and $g(t)$ is the Gaussian random force characterized by the following correlation.

$$< g(t_1)g(t_2) >= \frac{2a^2}{3\tau_e Z}\delta(t_1 - t_2) \tag{2.2}$$

3. Reptation
   The center of mass of each primitive path is forced to randomly move along the path with the diffusion constant $D_c = (1/3\pi^2)(a^2/\tau_e Z)$.

4. Constraint renewal
   If, as the result of the CLF and reptation, an end of a primitive path passes through the last slip link of the chain, the slip link and its partner are destroyed (constraint release). If, on the other hand, the length of a tail at the end of the path becomes longer than a, a new slip link is created at the end, and its partner is created on a randomly selected chain (constraint creation).

The stress per chain is calculated as follows.

$$\sigma_{\alpha\beta} = \sum_j F_{j\alpha} r_{j\beta}, \tag{2.3}$$

where $\mathbf{F}_j$ is the tension on the j-th subchain vector $\mathbf{r}_i$. We assume that the magnitude of the tension $|\mathbf{F}_j|$ is uniform along the primitive path (i.e., independent of j) and given by

$$F = \frac{3k_BT}{a}\frac{L}{L_{eq}}. \tag{2.4}$$

Then the stress contribution from a single chain is calculated as follows:

$$\sigma_{\alpha\beta} = \frac{3k_BT}{Za^2}\sum_j L\frac{r_{j\alpha}r_{j\beta}}{r_j}. \tag{2.5}$$

In the case of star polymers, we assume that the branch point does not move except for the macroscopic flow of the sample. The reptation is switched off, and the Rouse relaxation time of an arm is set to $\tau_R = 4\tau_e Z_a^2$, where $Z_a = M/M_e$ and M is the molecular weight of the arm. In this model, the motion of each arm is independent, and the rheological properties of star polymers do not depend on the number of arms.

2.2 Model parameter

In order to predict the rheological properties of an actual polymer, at least two parameters of the polymer should be known; one is the entanglement molecular weight M_e, and the other is the unit of time τ_e which is chosen to be the Rouse relaxation time of the chain with the molecular weight = M_e. M_e is used for obtaining $Z = M/M_e$ as an input for the simulation. In our slip-link model, M_e is related with the plataou modulus G_N by

$$G_N = \frac{4}{5}\frac{\rho RT}{M_e}, \tag{2.6}$$

where ρ is the density and R is the gas constant. The values of G_N for many polymers can be found in literature. In most literature, however, M_e is calculated from G_N by $M_e = \rho RT/G_N$. Thus, for example, M_e of polystyrene is not 18,000 (the commonly used value) but (4/5)18,000 $\sim$ 14,400.

The dimensionless stress s given by the simulation is related with the actual stress σ by

$$\sigma = G_e s, \tag{2.7}$$

where the stress scale G_e is related with the platau modulus G_N by

$$G_e = \frac{4}{15} G_N. \tag{2.8}$$

Since the platau modulus G_N is related with M_e by Eq.(2.6), it is in principle possible to determine the stress scale G_e from M_e. In actual simulations, however, we regard τ_e, G_e and M_e as independent parameters, and determine their values by fitting the simulation results to the experimental results for a particular reference sample with known molecular weight distribution under one deformation type (e.g., shear flow). Then the same values are used in predicting the rheological properties under different deformation types (such as uniaxial elongation), or for the samples with different molecular weight distributions.

Chapter 3

Starting PASTA

3.1 Outline of simulations

Simulations by PASTA usually take the following three steps:

1. Equilibration (or thermalization)
   Before starting the measurements, the initial structure should be equilibrated. The equilibrated structure is written into the Output UDF file, and it can be used for another simulation as a restarting file.

2. Measurements (steady flow and step deformation)
   Steady flow or step deformation of various types (shear, uniaxial elongation, etc.) is applied to the system, and the resulting stress components (shear stress, normal stress, etc.) are measured as functions of time.

3. Analysis
   From the measured stress components, we can obtain various rheological properties such as the steady flow viscosity, stress growth functions after start up of the flow, and the stress relaxation modulus. The dynamical moduli are obtained as the Fourier transform of the stress relaxation modulus.

FORK is a program to create an Input UDF file of PASTA. You can generate various molecular weight distributions by using FORK. You can also specify the simulation conditions for PASTA and create an Input UDF file for PASTA. In the following, a typical procedure of operating FORK and PASTA is explained.

3.2 PASTA Tutorial

Files

The files used or explained in this tutorial are listed below.

- Using FORK to create an Input UDF for PASTA

```
(.exe)        /PF_ENGINE/bin/$(PF_ENGINEARCH)/fork(.exe)
(Input UDF)   /PF_ENGINE/PASTA/FORK-1.4.1/sample/in_fork.udf
(Out def UDF) /PF_ENGINE/udf/forkdef.udf
(Output UDF)  /PF_ENGINE/PASTA/FORK-1.4.1/sample/out/out_fork.udf
```

- Calculation by PASTA

```
(.exe)        /PF_ENGINE/bin/$(PF_ENGINEARCH)/pasta(.exe)
(Input UDF)   /PF_ENGINE/PASTA/PASTA-2.7/sample/out_fork.udf
(Out def UDF) /PF_ENGINE/udf/PASTAout.udf
(Output UDF)  /PF_ENGINE/PASTA/PASTA-2.7/sample/out/out_pasta.udf
(Summary UDF) /PF_ENGINE/PASTA/PASTA-2.7/sample/out/sum_pasta.udf
```

```
(python)      pasta_python.py  pasta_stress.py  pasta_for_gt2gw.py
(Action)      pastaplot.act
```

- Analysis tools

```
(.exe)    /PF_ENGINE/PASTA/tools/bin/$(PF_ENGINEARCH)/gt2et(.exe)
                                                      gt2gw(.exe)
                                                      smooth(.exe)
(script)  /PF_ENGINE/PASTA/tools/bin/$(PF_ENGINEARCH)/gwsmooth
(sample)   z10.gt(for gt2et and gt2gw)   z10.fft(for gwsmooth)
```

`$(PF_ENGINEARCH)` specifies the OS type you are using.

Using FORK to create an Input UDF for PASTA

Let us simulate the steady shear viscosity of the polystyrene having the weight average molecular weight $M_w = 200,000$ and the polydispersity index $M_w/M_n = 2.5$.

1. Input UDF
   "`in_fork.udf`"

   Sample: Polystyrene $M_e = 14480$
   We set τ_e to -1, which means the time is measured not in unit of second but in unit of τ_e.
   Molecular weight distribution: log-normal distribution with $M_w/M_n = 2.5$.
   1000 chains are divided into 10 molecular weight components.
   Flow condition: shear flow
   `StrainRate` is set to $0.1/\tau_e$ (not $0.1s^{-1}$).

```
Name            M0                 Me     GN0   tau_e     MaxStretchRatio
PS             104              14480     0.2    -1.0                 4.4

ChainType       Mw         Mw_over_Mn     DistributionFunction
linear     200,000                2.5     LogNormal

Type_of_fraction         Num_of_Chain     NumDiv             DivType
Num_of_Chain                    1,000          9             Log

FlowType  DeformationType  StrainRate     dt    MaxTimeStep  IntervalStep
noflow    --                      0.0    1.0            100             1
flow      shear                   0.1    1.0            100             1
```

Figure 3.1: Editor window (tree view)

2. Output UDF
   FORK will create an Output UDF named "`out_fork.udf`". It can be used as an Input UDF for PASTA.

3. Running FORK
   You can run FORK either from GOURMET or from the command line.

   ♢ Running FORK from GOURMET

   (a) Read the Input UDF into GOURMET
   In GOURMET, select Open from the File menu, and open the Input UDF "`in_fork.udf`".

   (b) Engine Run window

   i. Select Engine Run from Tool menu. The Engine Run window will open. Select '`FORK`' from the engine list in the window.

   ii. In Input UDF:, enter "`in_fork.udf`".

   iii. In Output UDF:, enter "`out_fork.udf`".

   iv. In Working Dir:, enter the name of the directory in which the engine should be run. The specified directory will be created, and the Input UDF "`in_fork.udf`" will be copied into it. Do not specify the directory which contains the original Input UDF.

   (c) Run FORK
   Push the RUN button. Engine Control window will open and the calculation starts.

   (d) Viewing the Output UDF
   When the calculation finishes, open the Output UDF by using Open command in File menu.

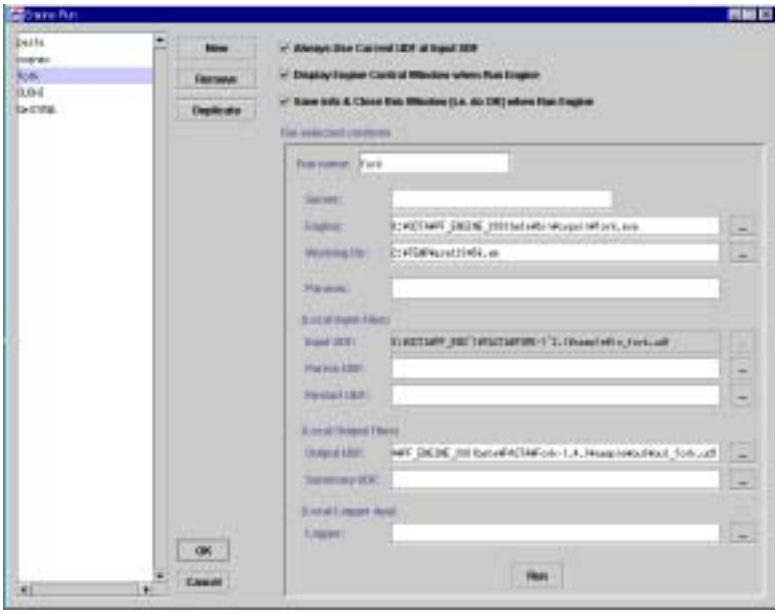

Figure 3.2: Engine Run window

♢ Running FORK from the command line
You can also run FORK by typing the following command after the shell prompt

```
% fork -I in_fork.udf -O out_fork.udf
```

4. Plot the Molecular Weight Distribution
You can plot the generated molecular weight distribution by using the plot function of GOURMET. Details are explained in 6.4.1.

Calculation by PASTA

1. Input UDF
The output of FORK, "`out_fork.udf`", will be used as the Input UDF for PASTA. If you have not generated this file, please use the one in `FORK-1.4.1/sample/out`.

2. Output UDF
The results will be written into the Output UDF "`out_pasta.udf`".

3. Running PASTA

♢ Running PASTA from GOURMET

(a) Read the Input UDF
Open the Input UDF "`out_pasta.udf`" in GOURMET window.

(b) Select PASTA in the Engine Run window

i. Select Engine Run from Tool menu. Then select '`PASTA`' from the engine list in the Engine Run window.

ii. Enter "`out_fork.udf`" in Input UDF:.

iii. Enter "`out_pasta.udf`" in Output UDF:.

iv. Enter "`pasta_summary.udf`" in Summary UDF:.

v. Enter the name of the directory in which the engine should be run. The specified directory will be created, and the Input UDF will be copied into it. Do not specify the directory which contains the original Input UDF.

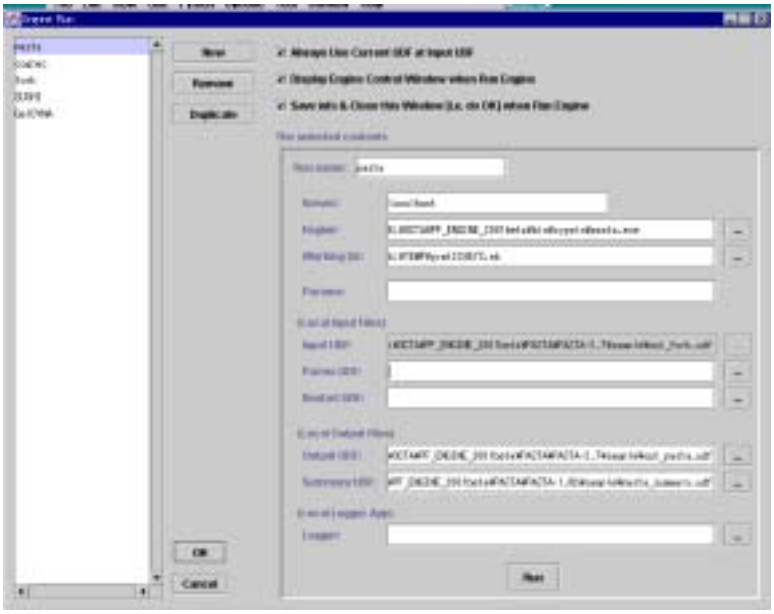

Figure 3.3: Engine Run window

(c) Run PASTA
Push the RUN button. Then Engine Control window opens and the calculation starts. You can use the following buttons in this window.

Pause :	Pause the job
Resume :	Restart the pausing job
Stop :	Stop the job, and output the current configuration
Kill :	kill the job

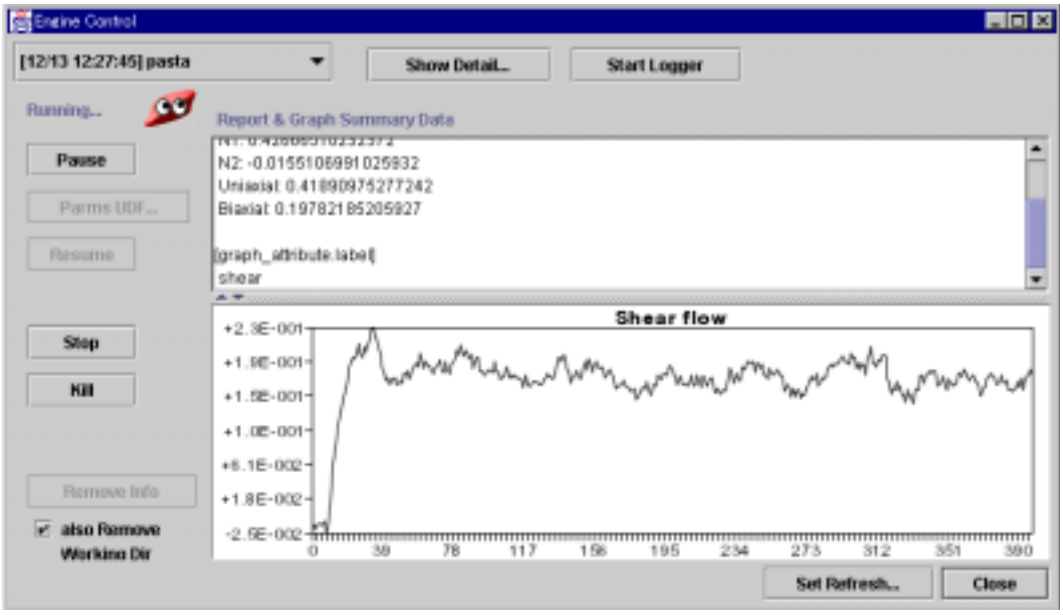

Figure 3.4: Engine Control window

(d) Check the Output UDF
Open the generated Output UDF in GOURMET window, and look through it.

♢ Running from the command line
To run PASTA from the command line, go to the directory where the Input UDF is, and type the following after the shell prompt:

```
% pasta -I out_fork.udf -O out_pasta.udf
```

4. Plotting the calculation results
The results of the calculation can be plotted by either of the following two methods:

- Using Action command
  (a) Read the Output UDF
  Open the Output UDF in GOURMET window.
  (b) Execute Action
  Right-click on the Output UDF name in the Tree panel. List of available Action commands will pop up (Fig.3.5). The following Action commands are prepared for PASTA.
  – plot_Stress
  Plot the time dependence of the stress (Fig.3.6).
  – plot_Viscosity_or_RelaxationModulus
  Plot the time dependence of the viscosity or the relaxation modulus.

Figure 3.5: Executing Action command

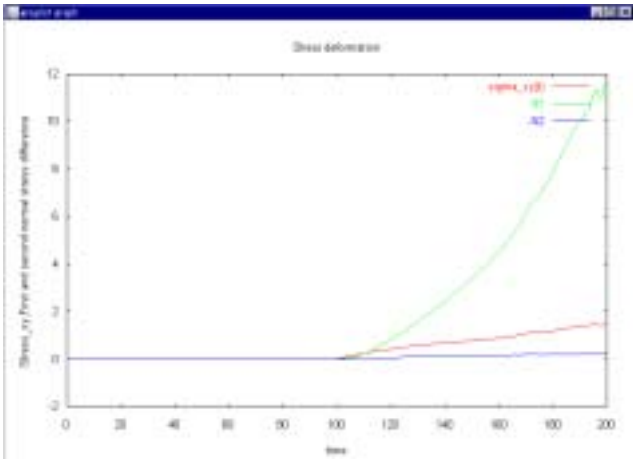

Figure 3.6: Example plot created by plot_stress

- Using Python script
  (a) Read the Output UDF
  Open the Output UDF in GOURMET window.
  (b) Load and execute the Python script
  Push the Load button, and load the Python script ‘‘pasta_python.py’’. Then push the Run button to execute the script. The data for plot will be created in the GraphSheet(Fig.3.8).

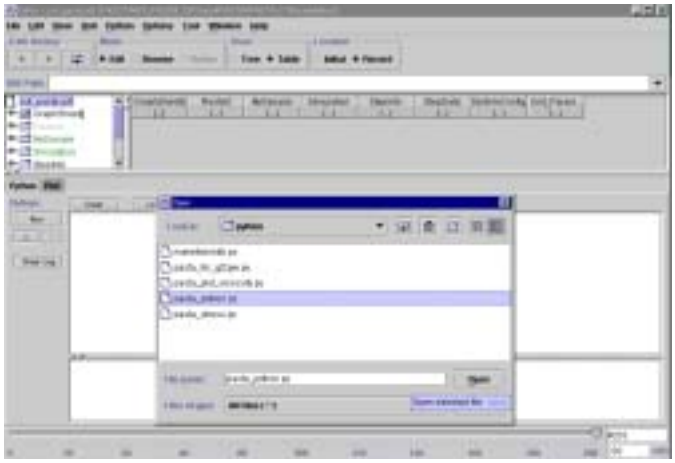

Figure 3.7: Load Python script “pasta_python.py”

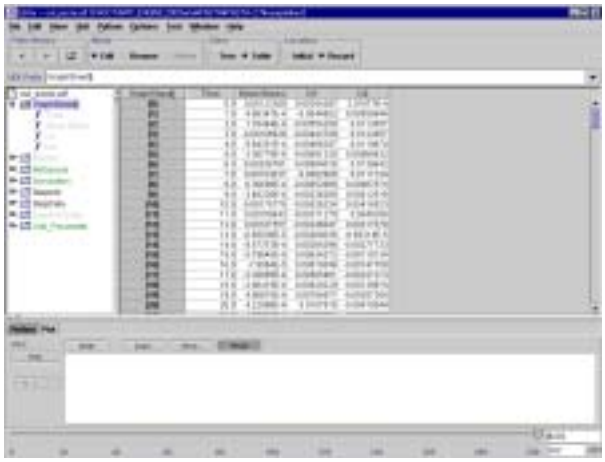

Figure 3.8: GraphSheet

  (c) Execute Make
  Select the GraphSheet, and push the Make button in the Plot panel. A script for gnuplot, such as shown below, will be automatically generated in the Plot panel, and a text file “plot.dat”, which contains the same data as GraphSheet, will be created in the GOURMET starting directory.

  Script for gnuplot

```
# plot command template for platform
# datafile name is fixed as "plot.dat" in the current version
set title "GraphSheet[]"
plot "plot.dat" using 1:2 title 'Time' with lines ,
     "plot.dat" using 1:3 title 'Shear Stress' with lines ,
```

```
"plot.dat" using 1:4 title 'N1' with lines ,
"plot.dat" using 1:5 title 'N2' with lines
```

(d) Plot the data
For example, if you want to plot the `Shear Stress` as a function of `Time`, modify the above script as follows:

```
# plot command template for platform
# datafile name is fixed as "plot.dat" in the current version
set title "GraphSheet[]"
plot "plot.dat" using 2:3 title 'Shear Stress vs. Time' with lines
```

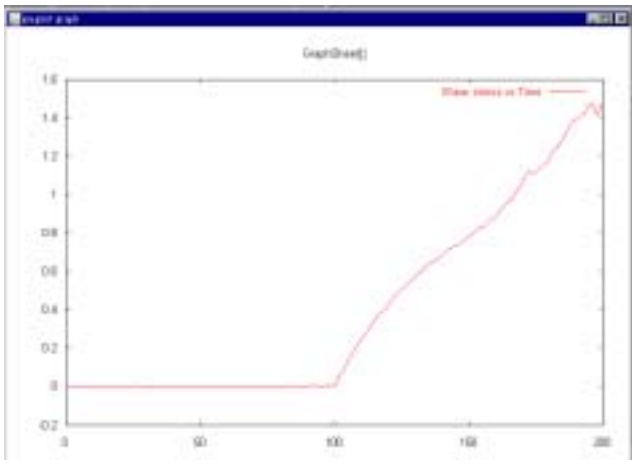

Figure 3.9: Plot of shear stress vs. time created by Python script ‘‘`pasta_python.py`’’

Then push the Plot button to display the plot. You can use other gnuplot commands in the script, if necessary. You can also read the text file “`plot.dat`” into a spreadsheet software. If you want to plot the viscosity, use the Python script “`pasta_plot_viscosity.py`”.

5. The Python script for data analysis
The Python script “`pasta_for_gt2gw.py`” can be used for extracting data from the Output UDF and creating an input data for analysis programs (`gt2et`, `gt2gw`).
The analysis programs are explained in Chapter 7.

Restart

1. Input UDF
The Input UDF is “`restart.udf`”. In this UDF, the results of the previous calculation, “`out_pasta.udf`”, is specified as the `Restart.UDFname`. If you have not created “`out_pasta.udf`”, use the one in `PASTA-2.7/sample/out`. Refer to section 7.1 for how to set `Restart.UDFname` by using Action command set_Restart_file.

```
FlowType  DeformationType  StrainRate      dt    MaxTimeStep  IntervalStep
flow      biaxial               0.05     1.0            100             1
```

2. Output UDF
Output UDF name is “`restart_out.udf`”.

3. Execute PASTA

(a) Read the Input UDF
Open the Input UDF "`restart.udf`" in GOURMET window.

(b) Check that `Restart.UDFname` is correctly set [1].

(c) Run PASTA either from GOURMET or from the command line. See page 10 for detail.

[1] `UDFname` may be either an absolute pathname, or a filename relative to the working directory.

Chapter 4

Sample problems

This chapter presents several sample calculations using PASTA. The files explained in this chapter are in the following directory:

`PF_ENGINE/PASTA/PASTA-2.7/sample`

4.1 Test sample

Let us simulate the shear flow of a monodisperse polymer.

Input sample file name	"`pasta_test_in.udf`"
Output sample file name	"`pasta_test_out.udf`"

100 chains of $Z = 10$ are prepared in the system. The value of `MaxStretchRatio` approximately corresponds to Polystyrene.

Calculation conditions

```
Z =10, MaxStretchRatio =4.4, NumChains =100
```

Equilibration (`FlowType` = 'noflow') of the system is carried out just for 100 steps. For real calculations, much longer equilibration is necessary (see the next section). Then, shear flow (`FlowType` = 'flow', and `DeformationType` = 'shear') is applied for 100 steps. `IntervalStep` = 10 indicates that the stress should be output at every 10 step interval. See to section 5.2 for more details of the parameters.

FlowType	DeformationType	StrainRate	dt	MaxTimeStep	IntervalStep
noflow	--	0.0	1.0	100	10
flow	shear	0.01	1.0	100	10

4.2 Shear viscosity

The rate dependence of the shear viscosity of several monodisperse linear polymers is shown in Fig.4.1. The number of chains was 100 for all the cases, the `MaxStretchRatio` was set to 4.4, the time step was $dt = 1.0$, and the number of steps for the equilibration and measurement are shown in the following table.

Z	Number of chains	Thermalization(steps)	shear flow(steps)
5	100	1000	10000
10	100	10000	100000
20	100	50000	500000
30	100	100000	1000000
60	100	1000000	10000000

Example Input and Output UDF for $Z = 10$ system can be found in the following files:

Input sample file name	"`pasta_shear_in.udf`"
Output sample file name	"`pasta_shear_out.udf`"

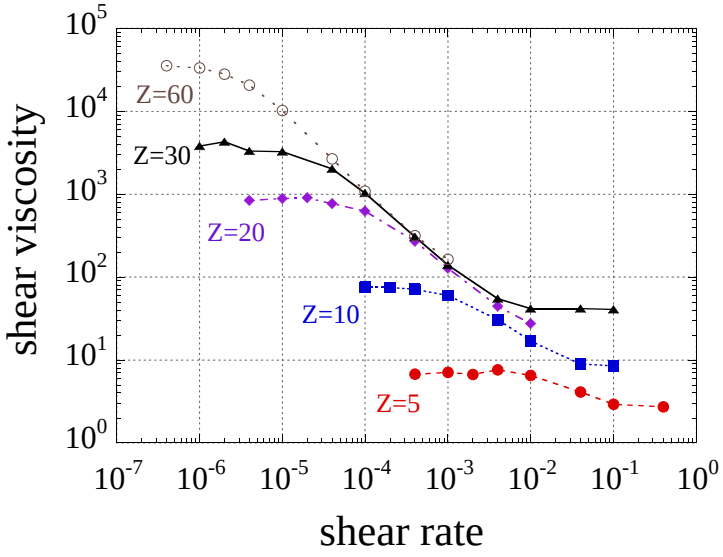

Figure 4.1: Shear rate dependence of the shear viscosity

4.3 Relaxation modulus

Relaxation modulus $G(t,\gamma)$ of the monodisperse linear polymer ($Z = 20$) is shown in Fig.4.2. $G(t,\gamma)$ is obtained by dividing the shear stress by the shear strain. The horizontal axis is the time after the application of the step shear strain. For $\gamma \leq 0.5$, $G(t,\gamma)$ is insensitive to γ and coincides with the linear relaxation modulus $G(t)$. An example of the sample data used here is shown below.

Input sample file name	"`pasta_step_mono_in.udf`"
Output sample file name	"`pasta_step_mono_out.udf`"

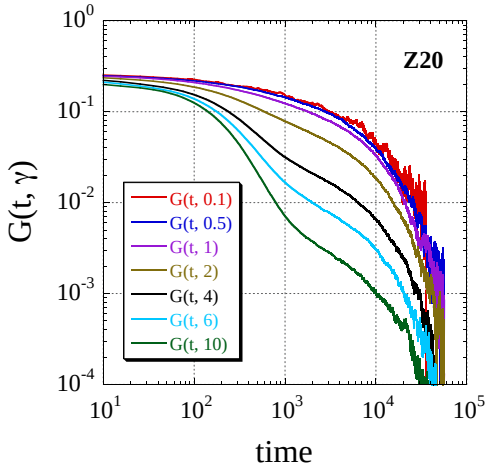

Figure 4.2: Relaxation modulus of the monodisperse linear polymer ($Z = 20$)

Calculation conditions

```
Z =20, MaxStretchRatio =4.4, NumChains =10000
```

```
FlowType  DeformationType  Strain     dt   MaxTimeStep   IntervalStep
noflow    --                  0.0    1.0        100000            100
step      shear               0.5    0.0             0              0
noflow    --                  0.0    1.0        100000              1
```

As an example of a polydisperse system, a system consisting of five molecular weight components was simulated. The calculation conditions are as follows.

Input sample file name "pasta_step_poly_in.udf"
Output sample file name "pasta_step_poly_out.udf"

Calculation conditions

```
Components          Z    NumChains    MaxStretchRatio
component-1      44.1           30                4.4
component-2      35.3          403                4.4
component-3      26.4         2743                4.4
component-4      17.6         5932                4.4
component-5       8.8          892                4.4

FlowType  DeformationType  Strain     dt   MaxTimeStep   IntervalStep
noflow    --                  0.0    1.0        100000            100
step      shear               0.5    0.0             0              0
noflow    --                  0.0    1.0        100000              1
```

The relaxation modulus of this system is shown in Fig.4.3 (step strain amplitude γ=0.5). Fig.4.4 shows the storage modulus G' and the loss modulus G'' of the same system. Here, G' and G'' are calculated as the Fourier transformation of the $G(t)$ data by using the analysis programs[1].

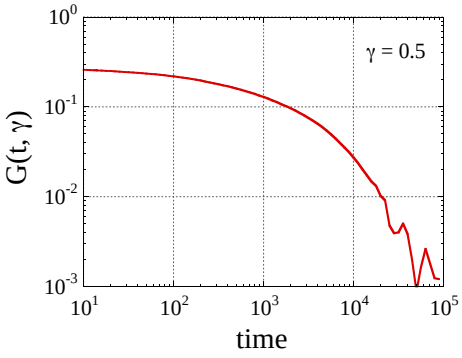

Figure 4.3: Relaxation modulus of the polydisperse system ($\gamma = 0.5$)

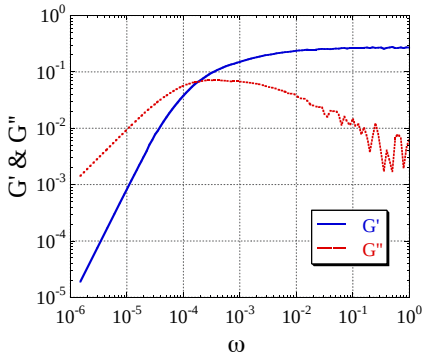

Figure 4.4: Storage and loss moduli(G', G'') of the polydisperse system

4.4 Elongational viscosity

The uniaxial elongational viscosity $\eta_E^+(t)$ of a monodisperse sample is shown in Fig.4.5. The uniaxial elongational stress($= N_1 + N_2/2$) is devided by the elongational rate to obtain $\eta_E^+(t)$. An example of sample data is shown below.

Input sample file name "pasta_unielong_in.udf"
Output sample file name "pasta_unielong_out.udf"

Calculation conditions

[1] refer to p.55.

```
Z =20, MaxStretchRatio =4.4, NumChains =10000

FlowType  DeformationType  StrainRate     dt   MaxTimeStep   IntervalStep
noflow    --                      0.0    1.0         20000             20
flow      uniaxial                0.001  1.0        100000              1
```

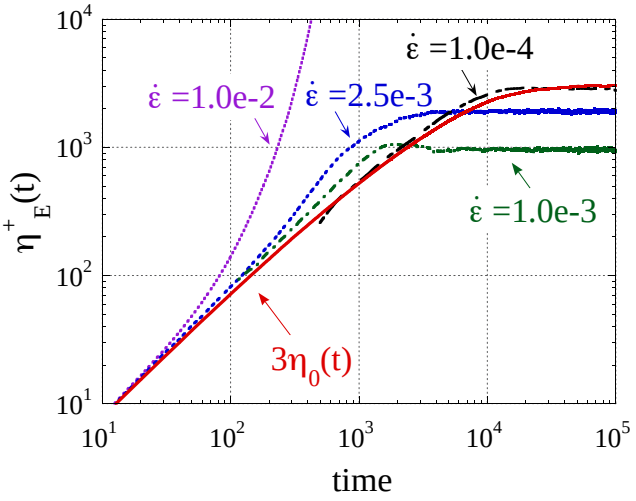

Figure 4.5: Elongational viscosity (uniaxial)

4.5 Shear viscosity of star polymers

The steady shear viscosities of the monodisperse linear polymers ($Z = 5, 10, 20, 30, 60$) and star polymers ($Z_a = 5, 10, 15$) are shown in Fig.4.6. Here, Z_a is the arm molecular weight M_a divided by the entanglement molecular weight M_e. Examples of the UDF files for star polymers used here can be found with the following names:

Input sample file name	"`pasta_star_in.udf`"
Output sample file name	"`pasta_star_out.udf`"

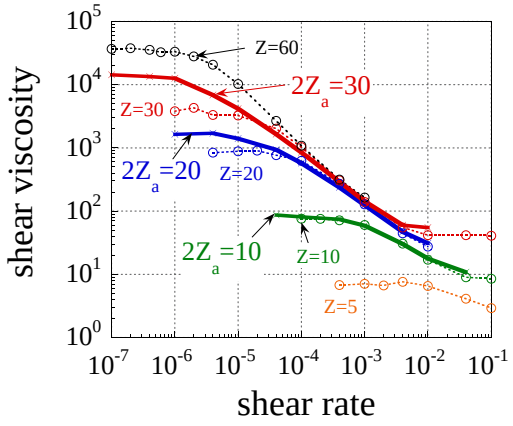

Figure 4.6: Steady shear viscosity of linear and star polymers

Calculation conditions

```
Za =10, MaxStretchRatio =4.4, NumChains =100

FlowType  DeformationType  StrainRate     dt   MaxTimeStep   IntervalStep
noflow    --                      0.0    1.0         10000            100
flow      shear                   0.01   1.0        100000             10
```

4.6 Notes

Equilibration of the system

In PASTA, it is necessary to relax the initial structure of the system before starting measurements. At least the `NumLinks`(average number of slip links) should become stationary before starting any measurements. More rigorously, the system should be equilibrated for several times longer than the longest relaxation time τ_d.

`NumChains` and `MaxTimeStep`

In the case of the steady flow (steady shear flow etc.), the number of chains (`NumChains`) can be small, 100 for example. If the strain rate is slower than $1/\tau_d$, the number of steps (`MaxTimeStep`) should be much longer than τ_d. If the strain rate becomes faster than $1/\tau_d$, `MaxTimeStep` much larger than $1/(strainrate)$ is sufficient.

In the case of the time dependent simulations (stress relaxation, elongational viscosity, etc.), the number of chains should be much larger, because the time average is not possible.

Unit system in PASTA

In the reduced unit system used by PASTA, time, molecular weight and stress are measured in unit of τ_e, M_e and $G_e = (4/15)G_N$, respectively. If you specify the actual values of τ_e, M_e and G_N in `Unit_Parameter`, the unit conversion can be easily done using GOURMET. See 'GOURMET Operation Manual' and section 5.2 for detail.

Chapter 5

Operation guide of PASTA

5.1 Execution

5.1.1 Editing Input UDF

This section explains the method of editing the Input UDF by GOURMET. Input UDF can also be directly edited by using a text editor. Some samples of Input UDF files can be found in
`PF_ENGINE/PASTA/PASTA-2.7/sample/`

1. Read the file to edit
   You can use the skeleton file "`PASTAin.udf`" as the starting point, or, use an Input UDF file of a previous calculation and modify it.

2. Edit the Input UDF

   (a) Sample
   Enter values for `Sample.Linear[]` and/or `Sample.Arm[]`. `Linear[]` is for the molecular weight components of linear polymers, and `Arm[]` for the arms of star polymers (in PASTA, the rheology of star polymers dose not depend on the number of the star arms.). You can also use Action command simpleFORK to generate a molecular weight distribution in `Sample`. See section 7.1 for Action.

   (b) Simulation
   Enter simulation conditions.

   (c) Save the file
   Save the file by using File/Save As. A new file name is required.

5.1.2 Running PASTA from GOURMET

1. Setup Environment Variable
   The Output UDF definition file is required for the execution of PASTA. The default name of the Output UDF definition file is "`PASTAout.udf`", and it should be located in the directory specified by the environment variable `UDF_DEF_PATH`. The OCTA installation program should have set the variable to `PF_ENGINE/udf`, and copied "`PASTAout.udf`" to that directory. If you encounter any trouble, please check whether `UDF_DEF_PATH` is correctly set and "`PASTAout.udf`" file exists in the directory `$(UDF_DEF_PATH)`.

2. Read the Input UDF
   In order to start a calculation, the Input UDF has to be loaded into GOURMET.

3. Register PASTA executable file (if not yet registered)

   (a) Select Engine Run from Tool menu. Engine Run window will open.

   (b) Push the New button, and enter 'PASTA' in the Run name:.

Figure 5.1: Editing Input UDF by GOURMET

(c) Enter the full path name of the PASTA executable file into the Engine: field. Use the ... button to locate the executable file (`pasta` or `pasta.exe`) . This file can usually be found in the directory `PF_ENGINE/bin/$(PF_ENGINEARCH)/`.

(d) Change the Working Dir:, if necessary. The engine will be run in this working directory. It should be different from the directory containing the original Input UDF. GOURMET will copy Input UDF into the working directory.

(e) Input UDF: field is automatically filled by GOURMET.

(f) Enter Output UDF name in Output UDF:. The Output UDF will be created by the engine in the working directory.

(g) Enter Summary UDF name in Summary UDF: (e.g. "`pasta_sum.udf`"). If Summary UDF: is empty, the Report & Graph Summary monitor function is not activated.

(h) Other fields can be left blank.

Example)

```
Run name:       PASTA
Engine:         PF_ENGINE/bin/pasta(.exe)
Input UDF:      (path name of the Input UDF. Set by GOURMET)
Output UDF:     pasta_test_out.udf
Summary UDF:    pasta_sum.udf
```

4. Starting PASTA
Push the Run button. Engine Control window will open, and the calculation will start. If the Summary UDF: was specified, current values of the stress will be reported, and a graph of the history of the stress will be shown. The interval, n_s, at which the stress is written to the Summary UDF is initially equal to the `IntervalStep` specified in the Input UDF. Everytime the number of data written to the Summary UDF exceeds 512, n_s is increased by a factor of 4, and the number of data in the history is reduced to $512/4 = 128$.

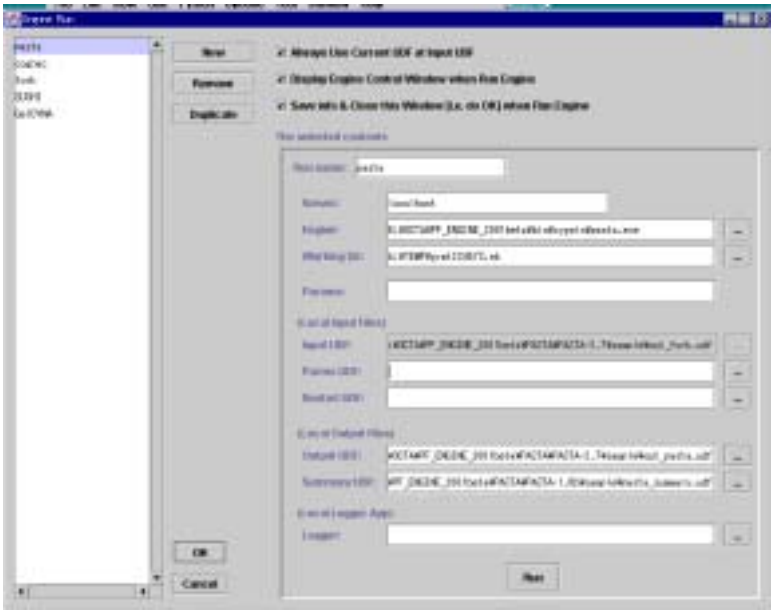

Figure 5.2: Engine Run window

The following buttons can be used for controlling the engine:

Pause :	Pause the job
Resume :	Restart the pausing job
Stop :	Stop the job, and output the current configuration
Kill :	kill the job

Figure 5.3: Engine Control window

5. Check the Output UDF
   Open the Output UDF in GOURMET window, and check its contents.

6. Plot the output data (Optional)
   In order to plot the stress components as a function of time, follow the instructions below:

(a) Load the Output UDF

(b) Load the Python script "`pasta_python.py`", and run it. The data for gnuplot will be created in the `GraphSheet`. The data is saved in a file named "`plot.dat`".

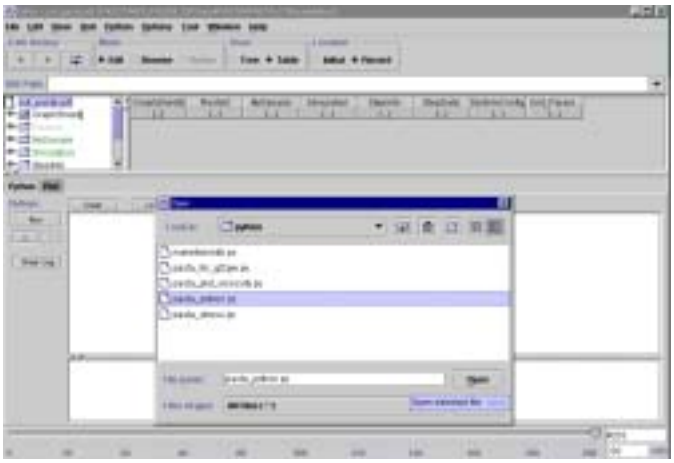

Figure 5.4: Loading Python script "`pasta_python.py`"

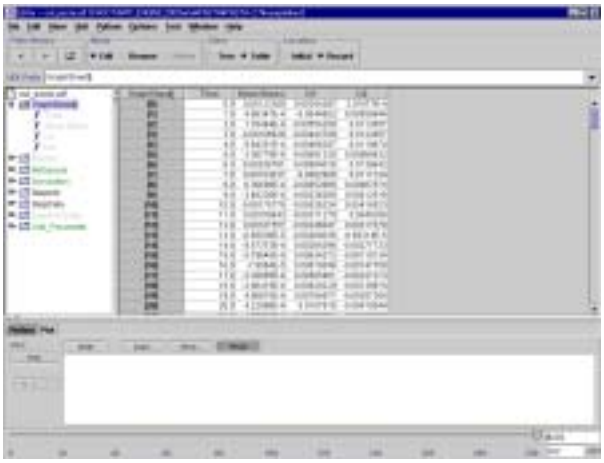

Figure 5.5: `GraphSheet`

(c) Execute Make
If the `GraphSheet` is displayed, push the Make button in the plot window. A script for gnuplot will be created automatically:

Example)

```
# plot command template for platform
# datafile name is fixed as "plot.dat" in the current version
set title "GraphSheet[]"
plot "plot.dat" using 1:2 title 'Time' with lines , \
     "plot.dat" using 1:3 title 'Shear Stress' with lines , \
     "plot.dat" using 1:4 title 'N1' with lines , \
     "plot.dat" using 1:5 title 'N2' with lines
```

If you want to plot the '`Shear Stress`' as a function of '`Time`', modify the script as follows:

```
# plot command template for platform
# datafile name is fixed as "plot.dat" in the current version
set title "GraphSheet[]"
plot "plot.dat" using 2:3 title 'Shear Stress' with lines
```

Here, # means the start of a comment.
The text file "plot.dat" is in the GOURMET starting directory. It can be read into common spreadsheet software.

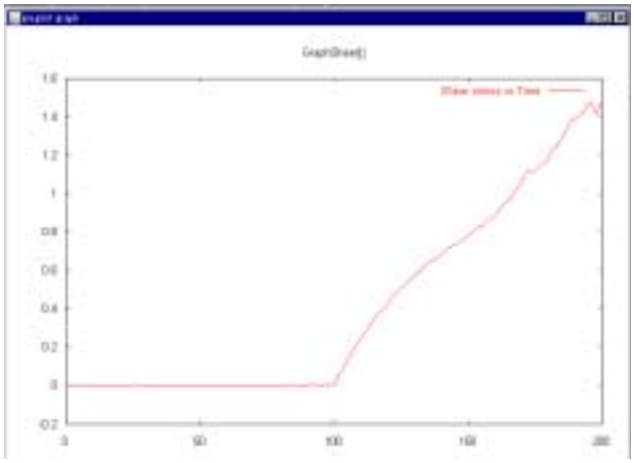

Figure 5.6: Example of a plot created by Python script "pasta_python.py"

5.1.3 Running from a shell prompt

PASTA can also be run from a command prompt. To do that, go to the directory containing the Input UDF, and type the following after the shell prompt:

```
% pasta -I inputUDF  -O outputUDF
```

If PASTA does not start or any error occurs, please check that the environment variable UDF_DEF_PATH is correctly set and "PASTAout.udf" is in the directory $UDF_DEF_PATH. If you are using Cygwin, also check that the environment variable CYGWIN_UDF_PATH is set to unix.

The progress of the simulation will be output to the logfile (default is the standard error). If you don't need the output, start pasta with an option -q.

5.2 UDF description

Two UDF files, "Input UDF" and the "Output definition UDF", are required to run PASTA. "Output definition UDF" for PASTA is "PASTAout.udf", and is installed in the directory PF_ENGINE/udf. If this directory is in UDF_DEF_PATH, it will be automatically searched and you don't need to specify it.

"Output UDF" is always newly created; appending to the existing Output UDF is not supported. Each Output UDF contains information of the system configuration at the end of the calculation. If the name of an Output UDF file is specified in the Restart.UDFname field of the Input UDF file of the next simulation, the simulation can be restarted from the configuration in the Output UDF.

The detailed structure of the Input UDF and Output UDF file is explained below.

5.2.1 Input UDF description

The Output UDF created by FORK can also be used as an Input UDF of PASTA. See the manual of FORK for detail. The information described in this section will be useful when you create Input UDF directly.

Two information should be specified in the Input UDF. One is the information about the sample to simulate, and the other is for the simulation procedure. For the information about the sample, either `Restart` or `Sample` must be specified.

```
//----- Restart -----

Restart: {
        UDFname: string    " name of restart UDF file"
}
```

Here, `UDFname` specifies the Output UDF file of the previous calculation. It may be the full pathname of the Output UDF file, or just the file name if the file is located in the working directory.

The `Sample` is defined as follows:

```
//----- Sample -----

class Component: {              // Molecular weight component
    Z: double[Me]               " Z = MW/Me"
    MaxStretchRatio: double     "Max Chain Stretch or Max Stretch ratio"
    NumChains: int              "number of chain"
};

Sample: {
    Linear[]: Component
    Arm[]: Component
};
```

`Linear[]` and `Arm[]` specify the molecular weight components of linear and star polymers, respectively. You can specify either `Linear[]` or `Arm[]`, or both. Each molecular weight component is represented by `Component`, whose elements are explained below:

```
    Z:
            (Molecular weight)/(Entanglement molecular weight)
            (positive real number)

    MaxStretchRatio:
            (Extended chain length)/(Equilibrium chain length)
            (0 or positive real number)

            If 0 is specified, the Gaussian chains which can extend
            without limit will be used.

    NumChains:
            The number of chains in this molecular weight component
            (positive integer).

            Specifying 0 does not cause an error, but the molecular weight
            component will be ignored.
```

You can specify either `Restart` or `Sample`, but not both.
The second information, the simulation procedure, will be specified in the following format:

```
//----- Simulation -----

class Deformation:{
    FlowType: select{
                    "noflow",
```

```
                    "flow",
                    "step"
    }                               "Type of Flow. PASTA Input condition"

    DeformationType: select{
                    "shear",
                    "uniaxial"
                    "biaxial"
                    "planar"
    }                               "Type of Deformation. PASTA Input condition."

    Strain: double                  "strain (only for FlowType==’step’)"
    StrainRate: double[1/Tau_e]     "strain rate. (only for FlowType==’flow’) "
    dt: double[Tau_e]               " TimeStep per 1 tau_e "
    MaxTimeStep: int                " number of iterations"
    IntervalStep: int               "output stress every IntervalStep steps"
}

Simulation: {
    Deformations[]: Deformation
};
```

`Simulation` is an array of `Deformation`. Each `Deformation` will be applied to the sample in the order it appears in `Simulation`. Meanings of each element of `Deformation` are described below:

```
    FlowType:
                    "noflow"    time evolution without flow.
                    "flow"      flow with fixed strain rate.
                    "step"      step strain is applied to the sample.
    DeformationType:
                    "shear"     shear
                    "uniaxial"  uniaxial elongation
                    "biaxial"   biaxial elongation
                    "planar"    planar elongation
                    (will be ignored if FlowType=="noflow")
    Strain:
                    amount of the step strain.
                    (will be ignored if FlowType is not "step")
    StrainRate:
                    strain rate (in unit of 1/tau_e).
                    (will be ignored if FlowType is not "flow")
    dt:
                    time step (in unit of tau_e).
                    A value around 1.0 is OK for usual cases.
    MaxTimeStep:
                    The number of iteration steps.
                    (will be ignored if FlowType=="step")
    IntervalStep:
                    The stress will be output at every IntervalStep steps.
                    (will be ignored if FlowType=="step")
```

PASTA-2.8 (or later) can output the data for the time evolution of the structure of a selected chain to a text file. The text file, which is called the trajectory file, can be used as an input file for the animation program `pastanim` (see section 7.4).

If you want to produce a trajectory file, please specify in the input UDF the data for `Trajectory` defined below:

```
//----- Trajectory -----

Trajectory: {
ChainID: int "index of the chain whose trajectory will be output"
Interval: int "output interval of the trajectory"
FileName: string "output file name"
}
```

Here, `ChainID` is the index of the chain whose trajectory you want to output. Suppose, for example, your `Sample` is as follows:

```
Sample: {
    Linear[]: [ // Z MaxStretchRatio NumChains
                { 20             0.0       200 } // Linear[0]
                { 30             0.0        50 } // Linear[1]
              ]
    Arm[]:    [
                { 10             0.0       100 } // Arm[0]
              ]
}
```

Then `ChainID` = 0 selects the first (or zero-th) chain in `Linear[0]`, `ChainID` = 200 selects the first chain in `Linear[1]`, and `ChainID` = 349 selects the last (99th) chain in `Arm[0]`. The trajectory will be output at every `Interval` time steps. `FileName` is the file name of the trajectory file.

The reduced unit system used in PASTA is described in the section surrounded by \begin{unit} and \end{unit}, and the actual values of the units are specified in `Unit_Parameter`.

```
\begin{unit}
const               =   0.2666666666  // 4/15
[Tau_e]             =   {$Unit_Parameter.tau_e}[s]
[Me]                =   {$Unit_Parameter.Me}[g/mol]
[GN0]               =   {$Unit_Parameter.GN0}[MPa]
[Stress]            =   [const*GN0]
\end{unit}

// Unit_Parameter
Unit_Parameter:{
    Name:string         "polymer name"
    tau_e:double[s]     "Unit of time "
    Me:double[g/mol]    "molecular weight between entanglements(g/mol)"
    GN0:double[MPa]     "plateau modulus GN0 (MPa)"
}
```

`[tau_e]`, `[Me]` and `[Stress]` are the unit of time, molecular weight and stress, respectively. Meanings of each element of `Unit_Parameter` are described below:

`Name:`	Polymer name or note.
`tau_e:`	Rouse relaxation time τ_e of a chain whose molecular weight = M_e. Unit of time.
`Me:`	Average molecular weight between entanglement points M_e (g/mol).
`GN0:`	Plateau modulus G_N^0 (MPa).

Note :

- Upper and lower case letters can be used in `FlowType` and `DeformationType`. For example, 'noflow', 'NoFlow', 'NOFLOW', etc. are all equivalent.
- If `MaxTimeStep` is not a multiple of `IntervalStep`, it will be increased to the next multiple of `IntervalStep`.

5.2.2 Output UDF description

Data output in the common section

For polydisperse system, the tractions having $Z <$ `Zcutoff` is ignored by PASTA (`Zcutoff` = 3.0 in the current version). Components with `NumChains` = 0 are also ignored. Thus the sample used in the simulation may be different from what specified in the Input UDF file. The actual sample used in the simulation is written as `MySample`, which is defined as follows:

```
//----- MySample -----

class MyComponent: {
Ctype: string "chain Type 'Linear' or 'Arm'"
Z: double[Me] "average number of slip link"
MaxStretchRatio: double "max stretch ratio of target polymer"
NumChains: int "number of chain"
WeightFraction: double "weight Fraction"
};

MySample: {
Zn: double[Me] "number-average Z(Average number of slip link)"
Zw: double[Me] "weight-average  Z(Average number of slip link)"
Zz: double[Me] "z-average Z(Average number of slip link)"
TotalChains: int "total number of chains in this sample"
NumComponents: int "number of MyComponents[]"
MyComponents[]: MyComponent
};
```

`MyComponent` is the information about each molecular weight component in the system. It contains `WeightFraction`, which is the weight fraction recalculated after disregarding the components having $Z <$ `Zcutoff`.

`MySample` consists of `MyComponents[]` and the following parameters:

```
Zn:
        Number average molecular weight divided by Me
Zw:
        Weight average molecular weight divided by Me
Zz:
        z-average molecular weight divided by Me
TotalChains:
        Total number of chains (an arm is counted as a chain)
NumComponents:
        The number of molecular weight components,
        i.e., the number of elements in the array MyComponents[]
```

`Deformation`

Whenever it starts new `Deformation`, the information of the `Deformation` is output as one record.

`Stress`

The information such as the current value of the stress is output as `StepInfo` and `StepData` into a single record of Output UDF. The data are output (1) at the start of the simulation, (2) at every `IntervalStep` steps, and (3) just after the application of a step strain.

`StepInfo` is defined as follows:

```
//----- StepInfo -----
```

```
StepInfo: {
Iteration: int "total iterations so far"
Time: double [Tau_e]  "current time"
Gamma: double "shear strain"
Eps: double "uniaxial (Hencky) strain"
};

    Iteration:
            The number of time steps counted from the beginning of this UDF.
            It is not increased by FlowType=="step".
    Time:
            current time = Iteration*dt
    Gamma, Eps:
            Strain applied so far.

Notes: All of these are reset to 0 when the simulation is restarted.
```

The `StepData` is defined as:

```
//----- StepData -----

StepData: {
TotalStress:    Stress          "total stress of the system"
SubStress[]:    Stress          "stress of each component"
SubData[]:      ComponentData   "length of each component, etc"
};

    TotalStress:
         Total stress of the system
    SubStress[]:
         Stress of each molecular weight component
    SubData[]:
         The average length of the chains of each molecular weight component, etc.
```

`SubStress[i]` is the stress if there is only the component i in the sample. `TotalStress` is given by the sum of `SubStress[i]`*(weight fraction of the component i).

The definition of `Stress` is as follows:

```
//----- Stress -----
class Stress: {
Shear:  double [Stress]     "shear stress"
N1:     double [Stress]     "1st normal stress diff. or  uniaxial stress"
N2:     double [Stress]     "2nd normal stress diff."
};
```

The unit of `Stress` is $3\rho RT/M_e = (15/4)G_N$.

When `DeformationType` is 'shear', each component of `Stress` has obvious meaning. In the case of various elongational flows, `Stress.Shear` has no meaning (averages to zero), while `N1` and `N2` have the following meanings:

- Uniaxial elongation

$$\begin{cases} v_x = \dot{\varepsilon}x & \sigma_E = \sigma_{xx} - \frac{1}{2}(\sigma_{yy} + \sigma_{zz}) \\ v_y = -\frac{1}{2}\dot{\varepsilon}y & \quad = N_1 + \frac{1}{2}N_2 \\ v_z = -\frac{1}{2}\dot{\varepsilon}z & \end{cases}$$

- Biaxial elongation

$$\begin{cases} v_x = \dot{\varepsilon}x & \sigma_B = \frac{1}{2}(\sigma_{xx} + \sigma_{yy}) - \sigma_{zz} \\ v_y = \dot{\varepsilon}y & \quad = \frac{1}{2}N_1 + N_2 \\ v_z = -2\dot{\varepsilon}z & \end{cases}$$

- Planar elongation

$$\begin{cases} v_x = \dot{\varepsilon}x & \sigma_{P_1} = \sigma_{xx} - \sigma_{yy} = N_1 \\ v_y = -\dot{\varepsilon}y & \sigma_{P_2} = \sigma_{zz} - \sigma_{yy} = -N_2 \\ v_z = 0 & \end{cases}$$

Here is the definition of `ComponentData`:

```
//----- ComponentData -----
class ComponentData:{
NumLinks: double "average number of slip links"
Length: double "average contour length"
};
```

```
NumLinks:
            Average number of slip links which exist at the present time
            on the chain belonging to this molecular weight component.
Length:
            Average contour length of the chain  belonging to
            this molecular weight component at the present time (unit a).
```

Information for a restart

The configuration of the system at the end of the calculation is output in the last record of the Output UDF as `SystemConfig`:

```
//----- system configurations (for restart) -----

class ChainConfig: {
Ctype: string "'linear' or 'arm'"
Z: double [g/mol] "Z (constant)"
LmaxInv: double "1/Lmax (constant)"
L: double "contour length"
Lt: double "tube length"
xf: double "tail length at the front of the tube"
xb: double "tail length at the back of the tube"
Rfirst: Vector3D "position of the first slip link"
NumLinks: int "number of slip links"
Links[]: LinkConfig "config. of each slip link"
};

SystemConfig: {
TotalChains: int "total number of chains"
NumComponents: int "number of components"
NumChains[]: int "number of chains in each component"
Chains[]: ChainConfig "config. of each chain"
};

//----- Vector3d -----

class Vector3d: {
    x:              double
    y:              double
    z:              double
}
```

An example record sequence

Suppose, for example, that the following deformations are applied to the sample:

```
Simulation: {
[
//FlowType  DeformType  strain   SRate    dt   MaxStep   Interval
{ "noflow"   ""           0.0    0.0     1.0       100       10 }
{ "step"     "shear"      1.0    0.0     0.0         0        0 }
{ "flow"     "shear"      0.0    0.001   1.0       100       10 }
]
}
```

Then the following records will be written into the Output UDF:

```
Record  0:   Stress in the initial state (t= 0)
Record  1:   Deformation[0] = { "noflow"  ""  0.0  0.0  1.0  100  10 }
Record  2:   Stress at t= 10
Record  3:   Stress at t= 20 ...
...
Record 11:   Stress at t= 100
Record 12:   Deformation[1] = { "step"  "shear"  1.0  0.0  0.0  0 0 }
Record 13:   Stress immediately after the step strain (t= 100)
Record 14:   Deformation[2] = { "flow"  "shear"  0.0  0.001  1.0  100  10 }
Record 15:   Stress at t= 110
...
Record 24:   Stress at t= 200
Record 25:   SystemConfig (for restart)
```

Chapter 6

- FORK - The support tool for PASTA

6.1 What is FORK ?

FORK is the support tool that generates an Input UDF for PASTA. It can generate various kinds of molecular weight distributions such as the Poisson distribution and Log-normal distribution. In addition, it can import GPC data from a text file.

In the input file for FORK, you should specify two information:

- Type of the molecular weight distribution you want to generate, and its parameter.
  For example, 'a Poisson distribution with $M_w = 25000$'.
- Deformations you want to apply to the system.
  For example, 'Steady shear flow with shear rate $= 0.01/s$'.

Then FORK will generate the molecular weight distribution, and create a UDF file, which can be used as an Input UDF for PASTA without modification.

6.2 FORK Overview

6.2.1 Generating molecular weight distribution

The following information is necessary for generating the molecular weight distribution by FORK.

- Weight-average molecular weight M_w
- The ratio of weight-average to number-average molecular weight M_w/M_n
- Type of distribution function
- The minimum molecular weight
- The maximum molecular weight
- The number of components
- The type of division

Type of distribution function

The molecular weight distribution of a polymer sample is usually determined by the polymerization mechanism. For example, the molecular weight distribution of the samples obtained by addition or condensation reaction is usually approximated by the Schultz-Zimm distribution, while the Poisson distribution is used for living polymerization, and the log-normal distribution is used for Ziegler method.

FORK can generate four distributions listed below. In the list, $n(M)$ is the number distribution function, $w(M)$ is the weight distribution function.

1. Schultz-Zimm distribution

$$n(M) = \frac{\beta^{\alpha+1}}{\Gamma(\alpha+1)} M^{\alpha} \exp(-\beta M) \tag{6.1}$$

$$\text{where} \quad M_n = \frac{\alpha+1}{\beta} \ , \quad M_w = \frac{\alpha+2}{\beta}$$

2. Poisson distribution

$$n(M) = \frac{1}{M_0} \frac{\nu^{\frac{M}{M_0}}}{\Gamma(\frac{M}{M_0}+1)} \exp(-\nu) \tag{6.2}$$

$$\text{where} \quad \nu = \frac{M_n}{M_0}$$

3. normal distribution

$$n(M) = \frac{1}{\sqrt{2\pi}\sigma} \exp[-\frac{(M-\mu)^2}{2\sigma^2}] \tag{6.3}$$

$$\text{where} \quad \mu = M_n \ , \quad \sigma^2 = M_n(M_w - M_n)$$

4. log-normal distribution

$$w(M) = \frac{1}{\sqrt{2\pi}\sigma M} \exp[-\frac{(\ln M - \mu)^2}{2\sigma^2}] \tag{6.4}$$

$$\text{where} \quad \mu = \frac{\ln M_w + \ln M_n}{2}, \quad \sigma^2 = \ln M_w - \ln M_n$$

These distribution functions satisfy the normalization condition:

$$\int_0^\infty n(M)dM = \int_0^\infty w(M)dM = 1 \tag{6.5}$$

Besides these continuous distributions, FORK can also handle discrete molecular weight distributions such as a monodisperse or bidisperse system. It is also possible to import experimental GPC data into FORK.

The number of components

The continuous distributions such as the Poisson distribution are discretized, i.e., approximated by discrete distributions with many components. In FORK, we use the convention that the molecular weight components in the discretized distribution are numbered as $i = 0, 1, 2, \ldots, n$, where $n + 1$ is the total number of components.

The minimum and maximum molecular weights

These parameters specifies the minimum $M_{\min}$ and maximum $M_{\max}$ molecular weights which should be included into the distribution. Thus the 0-th component corresponds to the minimum molecular weight, and the n-th component to the maximum molecular weight.

You can specify -1 for either or both of $M_{\min}$ and $M_{\max}$. In this case, $M_{\min}$ and/or $M_{\max}$ will be automatically determined by FORK. In the case of the Schultz-Zimm and Poisson distributions, $M_{\min}$ and $M_{\max}$ are determined as shown in Fig.6.1. For example, $M_{\min}$ is determined so that the weight fraction of the molecules whose molecular weight is lower than $M_{\min}$ is 0.05 %. $M_{\max}$ is determined in a similar way.

In the case of the normal and log normal distributions, $M_{\min}$ and $M_{\max}$ are set at $\pm 3\sigma$ from the center of the Gaussian distribution. Thus, for example, $M_{\min} = \mu - 3\sigma$ for the normal distribution, and $\ln M_{\min} = \mu - 3\sigma$ for the log-normal distribution.

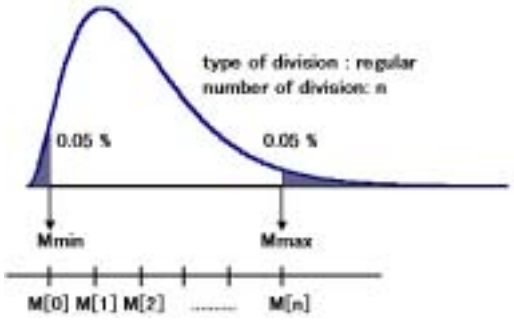

Figure 6.1: Determination of $M_{\min}$ and $M_{\max}$ for Shultz-Zimm and Poisson distributions.

The type of division method and the calculation of weight fraction

The discretized molecular weights, $M_i (i = 1, 2, \ldots, n)$, need not to be equally spaced. FORK can provide three types of divisions schemes: linear, log, and weight fraction.

If linear division scheme is used, the M_i's are equally spaced, and the weight fraction w_i of the component M_i is obtained by integrating the distribution function $w(M)$ from $(M_{i-1} + M_i)/2$ to $(M_i + M_{i+1})/2$, as shown in Fig. 6.2. If log-divisions scheme is used, M_i's are equally spaced on the $\ln M$ axis.

If the weight-fraction scheme is used, M_i's are determined such that $w_1 = w_2 = \ldots = w_{n-1} = 1/n$, and $w_0 = w_n = 1/(2n)$. See Fig. 6.3.

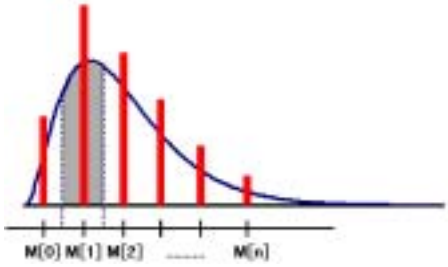

Figure 6.2: The weight distribution discretized by the linear division scheme.

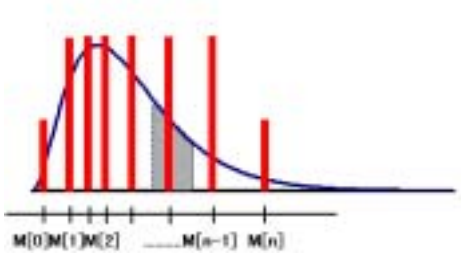

Figure 6.3: The weight distribution discretized by the weight fraction division scheme.

Recalculation of the average molecular weight

After the discretization, M_n and M_z are recalculated, because they have been slightly changed from the values you have given to FORK as inputs. The z-average molecular weight M_z is also calculated.

$$M_n = \sum_{i=0}^{n} M_i \cdot n_i = \frac{1}{\sum_{i=0}^{n} \frac{w_i}{M_i}} \tag{6.6}$$

$$M_w = \frac{\sum_{i=0}^{n} M_i^2 \cdot n_i}{\sum_{i=0}^{n} M_i \cdot n_i} = \sum_{i=0}^{n} M_i w_i \tag{6.7}$$

$$M_z = \frac{\sum_{i=0}^{n} M_i^3 \cdot n_i}{\sum_{i=0}^{n} M_i^2 \cdot n_i} = \frac{\sum_{i=0}^{n} M_i^2 w_i}{\sum_{i=0}^{n} M_i w_i} \tag{6.8}$$

Here, n_i and w_i are the normalized molar and weight fractions of the i-th component.

Importing GPC data

If you want to import your GPC data into FORK, please prepare your data in a text file with the following format.

Example: `gpc.dat`

```
20000           0.01
28000           0.03
45000           0.08
68000           0.19
90000           0.35
105000          0.15
130000          0.1
156000          0.03
180000          0.02
.....           .....
```

Here, the first column is the molecular weight, and the second column is the weight fraction.

Blending two or more samples

You can blend two or more molecular weight distributions. For example, you can blend a Poisson distribution with a monodisperse distribution. Blending method in FORK is very simple. If two samples having 10 and 5 components are blended, you will have a sample having 15 components.

6.2.2 Generating Input UDF for PASTA

Physical properties of the polymer

In the Input UDF for FORK, you must specify some physical properties of the polymer you want to simulate.

- `Name` : polymer name
- `M0` : molecular weight per monomer unit M_0 (g/mol) (for Poisson distribution only)
- `Me` : average molecular weight between entanglement points M_e (g/mol)
- `GN0` : plateau modulus G_N^0 (MPa)
- `tau_e` : unit of time (Rouse relaxation time of a chain whose molecular weight is M_e) τ_e (s.)
- `MaxStretchRatio` : maximum stretch ratio of the polymer chain λ_{max}
- `Temp` : standard temperature T (K)

Notes

- The only absolutely required parameter is M_e. For G_N^0, τ_e, λ_{max} or T, if the correct values are unknown, you can enter -1. Then the default values $G_N^0 = 1$, $\tau_e = 1$, $\lambda_{max} = 100$, and $T = 273$ will be used.
- If you specify τ_e in the unit of second, you must also specify the strain rates in the simulation condition in the unit of 1/second.
- You can find some physical properties for common polymers in the polymer database UDF file. See section 6.4.3.

Converting the molecular weight distribution for PASTA

The molecular weight M_i is converted into the average number of entanglements for the i-th components $Z_i = M_i/M_e$. Then the number of chains in this component is calculated from the molar fraction n_i and the total number of chains specified by the user.

Specifying the simulation condition of PASTA

Finally, you should specify the simulation condition of PASTA. You should enter one or more sets of the following information.

- `FlowType` : "flow", "noflow", or "step".
- `DeformationType` : "uniaxial", "biaxial", "planar", or "shear".
- `Strain` : magnitude of the step strain (only for flow type = "step").
- `StrainRate` : strain rate (only for flow type = "flow").
- `dt` : time step in units of τ_e.
- `MaxTimeStep` : number of iterations.
- `IntervalStep` : data output interval.

If τ_e was specified in the unit of second, then the strain rate should be specified in the unit of 1/second. If, on the other hand, τ_e was specified as -1, then the strain rate should be specified in the unit of $1/\tau_e$.

6.3 Sample calculations

This section explains some sample calculations using FORK.

The sample files described in this section can be found in the following directory:

```
PF\_ENGINE/PASTA/FORK-1.4.1/sample \\
```

6.3.1 Sample1

Polystyrene sample with $M_w = 200000$ and $M_w/M_n = 2.5$. A molecular weight distribution discretized into 10 components is generated. τ_e is input to be -1.0. This sets τ_e to a dimensionless value of 1.0. The simulation condition for PASTA is 100 steps of thermalization, followed by a shear flow with shear rate = $0.1/\tau_e$ for 100 steps.

Input sample name	"`in_fork.udf`"
Output sample name	"`out_fork.udf`"

Input data

- `PolymerData[]`

`Name`	PS
`Me:`	14480
`tau_e:`	-1.0 (use default)

- `MWDist.Sample[]`

`ChainType`	'linear'
`GPC_flag`	'off'
`Mw`	200000
`Mw_over_Mn`	2.5
`DistributionFunction`	LogNormal
`Type_of_fraction`	Num_of_Chain
`DivType`	Log
`NumDiv`	9 (10 components)
`Mmin`	-1
`Mmax`	-1 (automatically set by FORK)
`Num_of_Chain`	1000

- `Simulation.Deformation[0]`

`FlowType`	'noflow' (thermalization)
`Deformation`	'shear'
`Strain`	0.0
`StrainRate`	0.0
`dt`	1
`MaxTimeStep`	100
`IntervalStep`	1

- `Simulation.Deformation[1]`

`FlowType`	'flow'
`Deformation`	'shear'
`Strain`	0.0
`StrainRate`	0.1
`dt`	1
`MaxTimeStep`	100
`IntervalStep`	1

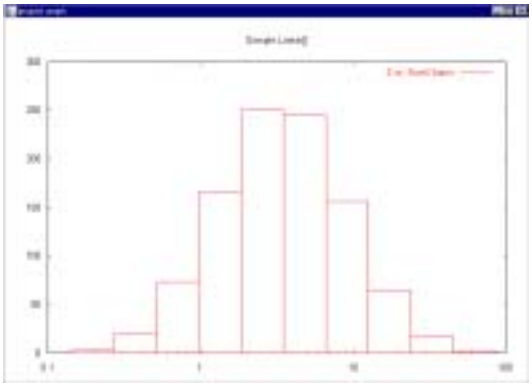

Figure 6.4: Molecular weight distribution of Sample1 (number of chains vs. log Z_i.)

Plot of the generated molecular weight distribution

Fig.6.4 shows the generated molecular weight distribution (number of chains vs. log Z_i). See section 6.4.1 for how to make the plot by using Python script and gnuplot.

6.3.2 Sample2

The molecular weight distribution is read from the GPC data file "`gpc.dat`".

Input sample name	"`gpcin_fork.udf`"
Output sample name	"`gpcout_fork.udf`"

Input data

- `PolymerData[]`

`Name`	PS
`Me`	14480
`tau_e`	-1.0 (use default)

- `MWDist.Sample[]`

`ChainType`	'linear'
`GPC_flag`	'on'
`GPC_file`	"`gpc.dat`"

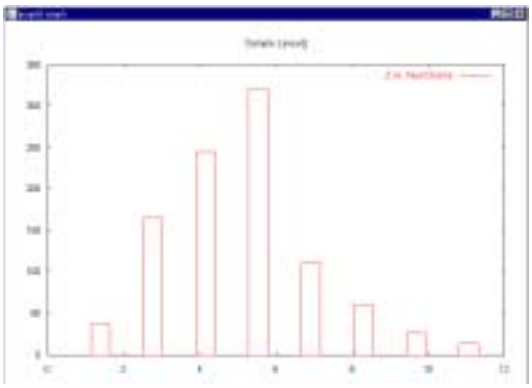

Figure 6.5: Molecular weight distribution of Sample2 (number of chains vs. Z_i).

Plot of the generated molecular weight distribution

Fig.6.5 shows the generated molecular weight distribution (number of chains vs. Z_i).

6.3.3 Sample3

Two samples of polystyrene are blended. The `Sample[0]` is a polydisperse sample with $M_w = 200000$ and $M_w/M_n = 2.5$. The other sample, `Sample[1]`, is a monodisperse sample with $M = 500000$. These two samples are blended with the weight fraction of 0.7 and 0.3.

Input sample name	"`blendin_fork.udf`"
Output sample name	"`blendout_fork.udf`"

Input data

- `PolymerData[]`

`Name`	PS
`Me`	14480
`tau_e`	-1.0 (use default)

- `MWDist.Sample[0]`

`ChainType`	'linear'
`GPC_flag`	'off'
`Mw`	200000
`Mw_over_Mn`	2.5
`DistributionFunction`	LogNormal
`Type_of_fraction`	WF
`WeightFraction`	0.7
`DivType`	Linear
`NumDiv`	10 (11 components)
`Mmin`	15000
`Mmax`	-1 (automatic)
`Num_of_Chain`	1000

- `MWDist.Sample[1]`

`ChainType`	'linear'
`GPC_flag`	'off'
`Mw`	500000
`Mw_over_Mn`	1.0
`DistributionFunction`	Discrete
`Type_of_fraction`	WF
`WeightFraction`	0.3
`DivType`	Linear
`Mmin`	15000
`Mmax`	-1 (automatic)
`Num_of_Chain`	1000

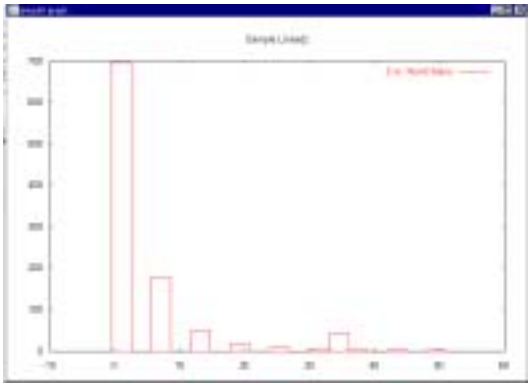

Figure 6.6: Molecular weight distribution of Sample3 (number of chains vs. Z_i).

Plot of the generated molecular weight distribution

Fig.6.6 shows the generated molecular weight distribution (number of chains vs. Z_i).

6.4 Operation guide of FORK

6.4.1 How to execute FORK

You can run FORK either from GOURMET or from the command line as a stand-alone program. In both cases, a UDF files "`forkdef.udf`" should exist in the directory specified by the environment variable `DEF_UDF_PATH`. GOURMET installation program should have set this variable to `PF_ENGINE/udf` and copied the UDF file to this directory. If FORK does not start correctly, check this environment variable and the UDF file.

Editing Input UDF

You can edit an Input UDF file for FORK either by GOURMET or by your favorite text editor. Here, an editing procedure by GOURMET is briefly described.

1. Open Input UDF
   Use Open in the File menu to open a UDF file by GOURMET. Sample UDF files can be found in `PF_ENGINE/PASTA/FORK-1.4/sample/`.

2. Edit Input UDF

   (a) Open polymer database UDF in another window
   Use New in the File menu to open a new window.
   Read `PF_ENGINE/POLYMERDATABASE/polymerdata.udf` into the new window(Fig.6.7). Find the name of the polymer you want to simulate. If the polymer has more than one `Properties`, determine the index of the `Properties` you want to use.

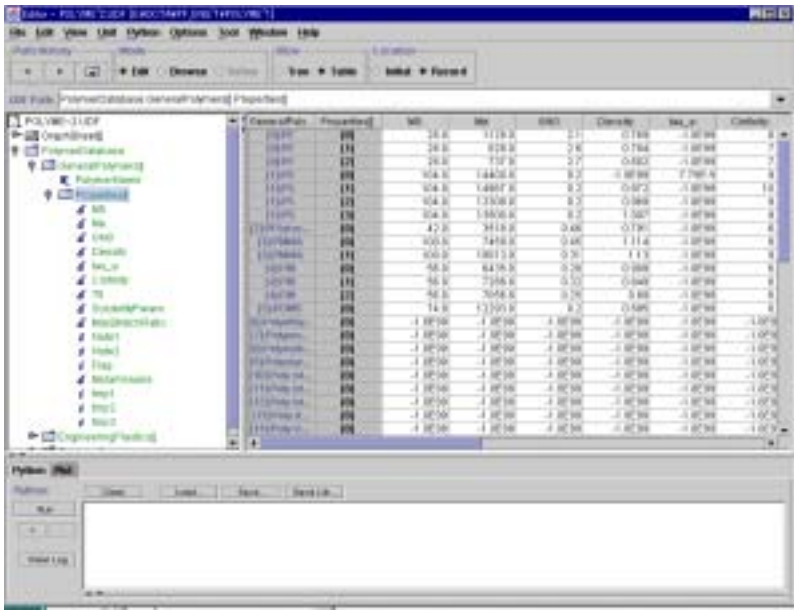

Figure 6.7: polymer data base UDF "`polymerdata.udf`"

   (b) Executing Action command set_PolymerData
   Set property data to `PolymerData` in Input UDF of FORK using Action command set_PolymerData on GOURMET. See section 7.1 for more detail of set_PolymerData.

   (c) Edit `MWDist`
   Use Insert Row in the Edit menu to add a new row to the `MWDist.Sample[]` array. Then enter appropriate values in the row.
   If you want to blend more than one samples, then add more rows to `MWDist.Sample[]`, and enter the total number of chains into `MWDist.TotalChain`.

(d) Edit Simulation
Add rows to Simulation.Deformations[], and enter the simulation conditions.

(e) Save
Save the Input UDF by Save As in the File menu (or by Save if it is OK to overwrite the file you have opened initially).

Running FORK from GOURMET

1. Open Input UDF
Open the Input UDF by GOURMET, if it has not been opened yet.

2. Register FORK in the Engine Run window
Use Tool/EngineRun command to open the Engine Run window. Push the New button to create a new entry in the list. Enter 'FORK' in the Run name: filed. In the Engine: field, enter the full path name of the FORK executable file; it can be found in PF_ENGINE/bin/$(PF_ENGINEARCH)/. You can use the ... button to locate the executable file. The Input UDF: field is automatically filled by GOURMET. Enter the Output UDF name in the Output UDF: field. Other fields can be left blank. The panel may look like this:

Example)

```
Run name:       FORK
Engine:         PF_ENGINE/bin/fork(.exe) (path name of the executable file)
Input UDF:      /somewhere/in_fork.udf (set by GOURMET)
Output UDF:     out_for.udf (will be created in the Working Dir)
```

3. Start FORK
Push the Run button. The Engine Run window will open. When FORK finishes, a string 'Stopped' will appear near the upper left corner of the window.

4. Check the Output UDF
Load the Output UDF into GOURMET, and check its contents. This Output UDF can be used as an Input UDF of PASTA without modification.

5. Plot the molecular weight distribution (optional)
You can plot the molecular weight distribution generated by FORK in the following way:

(a) Open the Output UDF
Open the Output UDF file, if it has not been opened yet.

(b) Select Sample.Linear[] or Sample.Arm[]
Click on Sample.Linear[] or Arm[] in the Tree panel (left hand side of the window). The data will be displayed in the right hand side of the window.

(c) Execute Make
Select Plot tab in the lower part of the window. If you push the Make button, GOURMET automatically generates a script for gnuplot as shown below:

Example)

```
# plot command template for platform
# datafile "plot.dat" is fixed current version
set title "Sample.Linear[]"
plot "plot.dat" using 1:2 title 'Z' with lines ,\
     "plot.dat" using 1:3 title 'MaxStretchRatio' with lines ,\
     "plot.dat" using 1:4 title 'NumChains' with lines
```

Please modify the script as follows:

```
# plot command template for platform
# datafile "plot.dat" is fixed current version
set title "Sample.Linear[]"
set boxwidth log(0.2)
set logscale x
plot "plot.dat" using 2:4 title 'Z vs. NumChains' with boxes
```

Push the Plot button. Gnuplot will be launched, and a plot will be created as shown in Fig.6.8). The data used by gnuplot is saved in a text file "`plot.dat`" in the GOURMET starting directory. You can read it into other plotting programs.

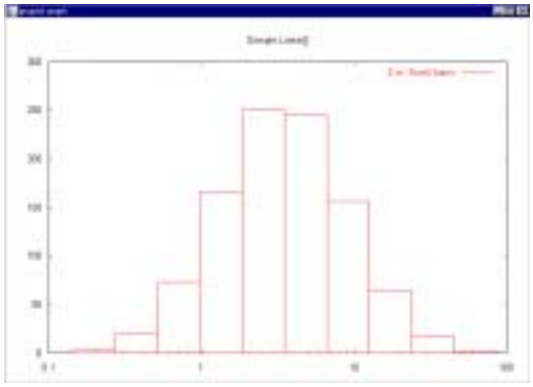

Figure 6.8: Plot of molecular weight distribution created by gnuplot

Running FORK from the command line

You can also run FORK as a stand-alone program. To do that, go to the directory containing the Input UDF, and type as follows:

`% fork -I` *Input-UDF-name* `-O` *Output-UDF-name*

6.4.2 Description of UDF

In this section, the structure of UDF files used or written by FORK is explained.

UDF list

The list of UDF files which FORK reads or writes.

- Definition UDF
  UDF in which the format of the Input and Output UDF are defined. Default name is "`forkdef.udf`".
- Input UDF
  UDF with input data.
- Output UDF
  UDF in which output data are written. It can be used as an input for PASTA.
- Polymer database UDF
  This UDF contains data of basic physical properties of some polymers.

Definition UDF

The default name of the definition UDF is "`forkdef.udf`". An Input or Output UDF file will contain the following line in its header.

```
#include forkdef.udf
```

Input UDF

Input UDF consists of the following three structures.

- `PolymerData`
  Basic physical parameters (such as the entanglement molecular weight M_e) of the polymer to be simulated.
- `MWDist`
  Type of the molecular weight distribution to be generated, and parameters of the distribution (such as the weight-average molecular weight M_w).
- `Simulation`
  Simulation conditions of PASTA. It is a sequence of deformations you want to apply to the sample.

Each data structure are described in the following:

PolymerData

```
PolymerData---Name
            |-M0
            |-Me
            |-GN0
            |-tau_e
            |-Temp
            |-MaxStretchRatio
```

- `Name` ... polymer name
- `M0` ... molecular weight of monomer unit M_0 (g/mol)
- `Me` ... molecular weight between entanglement points M_e (g/mol)
- `GN0` ... plateau modulus G_N^0 (MPa)
- `tau_e` ... Rouse relaxation time τ_e of a chain having the molecular weight = M_e.
- `T` ... standard temperature (K)
- `MaxStretchRatio` ... maximum stretch ratio of a polymer chain λ_{max}.

Notes

1. You can enter a value of -1 for `GN0`, `MaxStretchRatio` or `T`. In this case, default values $G_N^0 = 1$ (MPa), $\lambda_{max} = 100$, or $T = 273$ (K) will be used.
2. If you enter the value of `tau_e` in the unit of second, then the `StrainRate` in `Simulation.Deformations[]` should be entered in the unit of 1/second. If, on the other hand, -1 is entered for `tau_e`, then τ_e is used as the unit of time, and the `StrainRate` should be entered in the unit of $1/\tau_e$.

MWDist

`MWDist` has the following structures.

```
MWDist---Sample[]---ChainType
       |          |-GPC---GPC_flag
       |          |     |-GPC_file
       |          |-Mw
       |          |-Mw_over_Mn
       |          |-DistributionFunction
       |          |-Type_of_fraction
       |          |-WeightFraction
       |          |-Num_of_Chain
       |          |-NumDiv
       |          |-DivType
       |          |-Mmin
       |          |-Mmax
       |-TotalChain
```

- `TotalChain` ... The total number of chains in the blended sample. This is only necessary if `Sample[]` has more than one elements (i.e., if you blend two or more samples), and `Sample[].Type_of_fraction` is `Num_of_Chain`.

Sample[]

Each element of this array specifies the information necessary to generate a discretized molecular weight distribution.

- `ChainType` ... Enter 'linear' or 'arm', depending on whether the chain is a linear polymer or an arm of a star polymer.
- `GPC`
  - `GPC_flag` ... Select 'ON' or 'OFF'. If this flag is 'ON', the molecular weight distribution will be read from a text file.
  - `GPC_file` ... When `GPC_flag` is 'ON', specify the pathname of the text file which contains the GPC data.
- `Mw` ... Weight-average molecular weight
- `MW_over_Mn` ... The ratio of the weight-average molecular weight to number-average molecular weight M_w/M_n.
- `DistributionFunction` ... Type of the distribution function.

Type of distribution function	DistributionFunction
Normal distribution	'Normal' 'N' 'normal' 'n'
Log-normal distribution	'LogNormal' 'LN' 'lognormal' 'ln'
Schultz-Zimm distribution	'SchultzZimm' 'SZ' 'schultzzimm' 'sz'
Poisson distribution	'Poisson' 'P' 'poisson' 'p'
Discrete distribution	'Discrete' 'D' 'discrete' 'd'

  'Discrete' is used for a monodisperse sample. If you need a bidisperse (or multi-disperse) sample, you should blend two (or more) monodisperse samples.

- `Type_of_fraction` ... When blending more than one samples, the fraction of each sample in the blend can be specified in two ways. If `Type_of_fraction` is set to 'WeightFraction' or 'WF', then the weight fraction of each sample can be specified in `Sample[].WeightFraction`. If `Type_of_fraction` is set to 'Num_of_Chain' or 'NC', the number of chains of each sample is specified in `Sample[].Num_of_Chain`.

- `WeightFraction` ... Used only when `Type_of_fraction` is set to 'WeightFraction' or 'WF'.

  If there is only one `Sample`, Enter 1.0.
  Then `Sample[].Num_of_Chain` is determined from `MWDist.TotalChain` automatically.

  If two or more `Samples` are blended, Enter the weight fraction of each sample within the range 0.0 - 1.0.

- `Num_of_Chain` ... Used only when `Type_of_fraction` is set to 'Num_of_Chain' or 'NC'.

  Enter the number of chains in each sample.

- `NumDiv` ... number of divisions (n). The total number of molecular weight components is $n + 1$.

- `DivType` ... type of division.

Type of division	`DivType`
regular division	'Linear'
logarithm division	'Log'
weight fraction division	'WFraction'

- `Mmin` ... the minimum molecular weight. If `Mmin` is specified as -1, the `Mmin` value is automatically set by FORK.

- `Mmax` ... the maximum molecular weight. If `Mmax` is specified as -1, the `Mmax` value is automatically set by FORK.

Simulation
`Simulation` has the following structures.

```
Simulation---Deformations[]---FlowType
                            |-DeformationType
                            |-dt
                            |-Strain
                            |-StrainRate
                            |-MaxTimeStep
                            |-IntervalStep
```

Deformations[]
The deformations to be applied to the sample by PASTA.

- `FlowType` ... 'flow', 'noflow', or 'step'.

- `DeformationType` ... 'shear', 'uniaxial', 'biaxial' or 'planar'.

- `dt` ... simulation time step in the unit of τ_e

- `Strain` ... amount of step strain (only for `FlowType`=='step').

- `StrainRate` ... strain rate (only for `FlowType`=='flow').

  If `PolymerData.tau_e` is set to -1, `StrainRate` should be specified in the unit of $1/\tau_e$.
  If `PolymerData.tau_e` is specified in the unit of second, `StrainRate` should be entered in the unit of 1/second. In this case, FORK converts the strain rate into dimensionless number (i.e., in the unit of $1/\tau_e$), and write the converted value in the Output UDF, which will be read by PASTA.

- `MaxTimeStep` ... number of iterations

- `IntervalStep` ... stress is output at every `IntervalStep` steps.

Output UDF

Output UDF consists of the following three structures.

- `Results`
  Various average molecular weights of the sample calculated from the discretized distribution. In the case of blend, average molecular weights for the whole (blended) sample as well as those for each blend component are given.
- `Sample`
  The (blended) sample to be used by PASTA.
- `Simulation`
  Simulation conditions for PASTA.

Each data structure is described below.

Results

`Results` has the following structure:

```
Results---MWDistResults[]---AvMn
        |                  |-AvMw
        |                  |-AvMz
        |                  |-Mw_over_Mn
        |                  |-Mz_over_Mw
        |                  |-InputWFraction
        |-BlendedResults---AvMn
                         |-AvMw
                         |-AvMz
                         |-Mw_over_Mn
                         |-Mz_over_Mw
                         |-InputWFraction
```

MWDistResults[]

Various average molecular weights of each blend component. When `ChainType` is 'Arm', average molecular weights are calculated by considering each arm as a molecule.

- `AvMn` ... number-average molecular weight
- `AvMw` ... weight-average molecular weight
- `AvMz` ... z-average molecular weight
- `Mw_over_Mn` ... the ratio of `AvMw` to `AvMn`
- `Mz_over_Mw` ... the ratio of `AvMz` to `AvMw`
- `InputWFraction` ... the weight fraction of this blend component (copied from input)

BlendedResults

Various average molecular weights of the blended sample.

- `AvMn` ... number-average molecular weight
- `AvMw` ... weight-average molecular weight
- `AvMz` ... z-average molecular weight
- `Mw_over_Mn` ... the ratio of `AvMw` and `AvMn`
- `Mz_over_Mw` ... the ratio of `AvMz` and `AvMw`
- `InputWFraction` ... the sum of the weight fractions of the blend components.

Sample
`Sample` has the following structure:

```
Sample---Linear[]---Z
       |          |-MaxStretchRatio
       |          |-NumChains
       |-Arm[]------Z
                  |-MaxStretchRatio
                  |-NumChains
```

Linear[],Arm[]
Each element of `Linear[]` and `Arm[]` contains the information of one molecular weight component.

- `Z` ... the average number of slip links $Z = M/M_e$
- `MaxStretchRatio` ... maximum stretch ratio of a chain
- `NumChains` ... number of chains in this component

Simulation
`Simulation` has the following structure:

```
Simulation---Deformations[]---FlowType
                            |-DeformationType
                            |-dt
                            |-Strain
                            |-StrainRate
                            |-MaxTimeStep
                            |-IntervalStep
```

Deformations[]
The contents of `Deformations[]` are the same as that in the Input UDF.

6.4.3 Description of the polymer database UDF

The polymer database UDF consists of the following structures.

```
PolymerDatabase---GeneralPolymers[]---PolymerName
                 |                   |-Properties[]
                 |
                 |-EngineeringPlastics[]---PolymerName
                 |                        |-Properties[]
                 |
                 |-Rubbers[]---PolymerName
                             |-Properties[]


-------Properties[]---M0
                     |-Me
                     |-GN0
                     |-Density
                     |-tau_e
                     |-Cinfinity
                     |-Temp
                     |-SolubilityParam
                     |-MaxStretchRatio
                     |-Note1
                     |-Note2
                    ------------- below are not used by the polymer database.
                     |-Flag
                     |-MolarVolume
                     |-tmp1
                     |-tmp2
                     |-tmp3
```

Polymers are classified into the following three categories according to their properties.

- `GeneralPolymers[]`
  General polymers.
- `EngineeringPlastics[]`
  Engineering plastics are defined as thermoplastic resins which maintain the dimensional stability and mechanical properties at temperatures above 100°C or below 0°C. For example, polyacetal, polyamide, polyimide, polyester, polyether, polyether imide, polycarbonate, polysulfone etc...
- `Rubbers[]`
  Elastomers are defined as the amorphous polymer with the glass transition temperature lower than the room temperature, or the crystalline polymers with the melting point lower than the room temperature, both exhibiting rubber-like elasticity. For example, polybutadiene, polyisoprene, etc...

GeneralPolymers[],EngineeringPlastics[],Rubbers[]
Each category has the same structure.

- `PolymerName` . . . polymer name

Properties[]
Physical properties of each polymer. Note that many properties have a unit. If the data are not available for a property, $-1.0e^{-99}$ is to be entered.

- `M0` . . . molecular weight of the monomer unit (g/mol).

- `Me` ... molecular weight between entanglement point (g/mol).
- `GN0` ... plateau modulus (MPa).
- `Density` ... density (g/cm^3).
- `tau_e` ... Rouse relaxation time(second) of the molecule having the molecular weight = M_e.
- `Cinfinity` ... the characteristic ratio.
- `Temp` ... standard temperature (K).
- `SolubilityParam` ... solubility parameter (J/cm^3).
- `MaxStretchRatio` ... maximum stretch ratio of a chain.
- `Note1,Note2` ... The reference article from which the data are taken, etc.

The following are not used in the polymer database.

- `Flag` ... the flag for the calculation of χ-parameter. (See SUSHI reference manual.)
- `MolarVolume` ... molar volume (cm^3/mol).
- `tmp1, tmp2, tmp3` ... reserved for future use.

Chapter 7

Analysis tools

In this chapter, Action commands, Python scripts and stand-alone programs for the analysis of the input and output UDF of PASTA will be explained.

7.1 Action commands

The following Action commands[1] are available.

Action command	File Type	Location
simpleFORK	Input UDF of PASTA (Output UDF of FORK)	`Sample`
set_Restart_file	Input UDF of PASTA	`Restart`
set_GPC_file	Input UDF of FORK	`MWDist.Sample[].GPC_INPUT`
set_PolymerData	Input UDF of FORK	`PolymerData`
plot_Stress	Output UDF of PASTA	`output UDF name`
plot_Viscosity_or _RelaxationModulus	Output UDF of PASTA	`output UDF name`

These commands are defined in the following files:

Action command	Action file
simpleFORK	`PF_ENGINE/action/simplefork.act`
set_Restart_file	`PF_ENGINE/action/pastainput.act`
set_GPC_file	`PF_ENGINE/action/pastainput.act`
set_PolymerData	`PF_ENGINE/action/pastainput.act`
plot_Stress	`PF_ENGINE/action/pastaplot.act`
plot_Viscosity_or _RelaxationModulus	`PF_ENGINE/action/pastaplot.act`

Each Action command is explained below.

1. simpleFORK

   simpleFORK is an easy tool to regenerate the molecular weight distribution in a PASTA input UDF. To use simpleFORK :

   (a) Reading an Input UDF of PASTA
   Open an existing Input UDF of PASTA by GOURMET.

[1] See section 5.3 of GOURMET Operating manual for general information about Action.

(b) Execute simpleFORK
Right-click on `Sample` in the Tree panel, and select simpleFORK from the pop-up menu. A simpleFORK window will pop up (Fig.7.1). Enter suitable data into the Values column in the window. See Operation guide of FORK (section 6.4) for the meaning of each field. Note that you should enter the number of slip-links $Z = M/M_e$ instead of molecular weight M. If you push the OK button, the `Sample` will be regenerated.

(c) Check new data in `Sample`, and save this file.

Figure 7.1: simpleFORK window

2. set_Restart_file

set_Restart_file is an easy tool to set the restart file name in PASTA input UDF by 'mouse' select. To use set_Restart_file:

(a) Read Output UDF of PASTA
Open an Output UDF of PASTA by GOURMET.

(b) Execute set_Restart_file
Right-click on `Restart` in the Tree panel, and select set_Restart_file from the pop-up menu. set_Restart_file window will pop up (Fig.7.2). Right-click on the blank filed in the Values column, and select the restart file you want to use from the pop-up menu.

3. set_GPC_file

set_GPC_file is an easy tool to set the GPC data file name in FORK input UDF by 'mouse' select. set_GPC_file can be used in the same way as set_Restart_file.

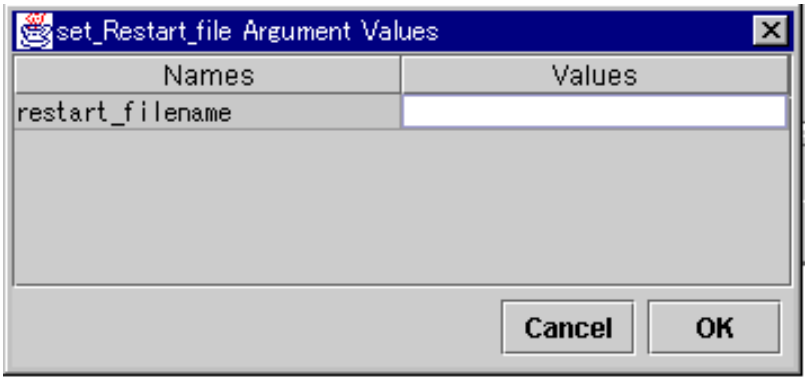

Figure 7.2: set_Restart_file window

4. set_PolymerData

set_PolymerData is an easy tool to set the PolymerData in Input UDF of FORK. To use set_PolymerData:

(a) Read Input UDF and "polymerdata.udf"
Open an Input UDF of FORK and "polymerdata.udf" by GOURMET. "polymerdata.udf" is usually located in PF_ENGINE/POLYMERDATABASE.

(b) Execute set_PolymerData
Right-click on PolymerData in the Tree panel, and select set_PolymerData from the pop-up menu. set_PolymerData window will pop up (Fig.7.3). In "polymerdata.udf", find POLYNAME and PROPERTIES_NUMBER of the polymer you want to simulate, and enter them in the set_PolymerData window. If you push OK button, data will be generated in the PolymerData of the input UDF.

(c) Check new data in PolymerData, and save this file.

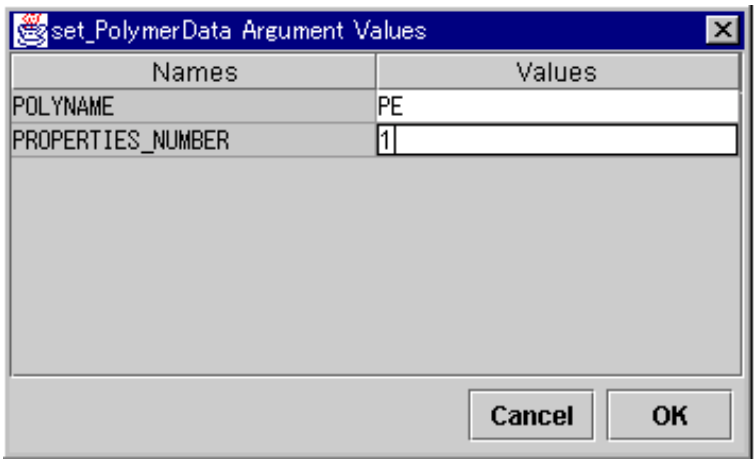

Figure 7.3: set_PolymerData window

5. plot_Stress and plot_Viscosity_or_RelaxationModulus

   plot_Stress plots the time dependence of the stress using gnuplot. plot_Viscosity_or_RelaxationModulus plots the time dependence of the viscosity or the relaxation modulus. To use these commands:

   (a) Read Output UDF
   Open an Output UDF of PASTA by GOURMET.

   (b) Execute plot_Stress (or plot_Viscosity_or_RelaxationModulus)
   Right-click on the Output UDF name at the top of the Tree panel, and select plot_Stress (or plot_Viscosity_or_RelaxationModulus) from the pop-up menu. Gnuplot graph window will pop up, and the suitable plot will be drawn in the window (for example, Fig.7.4).

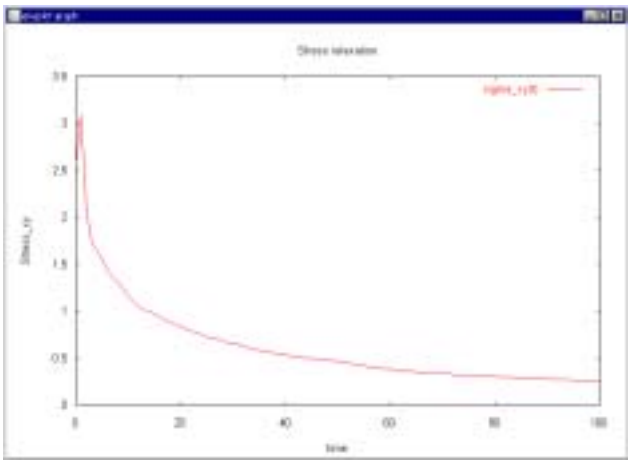

Figure 7.4: Gnuplot graph window created by plot_Stress command.

7.2 Python scripts

Following Python scripts are available.

`pasta_python.py`	plot stress vs. time
`pasta_plot_viscosity.py`	plot viscosity vs. time
`pasta_stress.py`	write stress data into a text file
`pasta_for_gt2gw.py`	create input files for analysis program gt2et and gt2gw

These scripts can be found in the following directory:

```
PF_ENGINE/PASTA/PASTA-2.7/python
```

Each script is explained below.

1. "`pasta_python.py`"

   This script should be run from the Plot panel of GOURMET. It plots the shear stress, first and second normal stress differences, etc. as a function of time using gnuplot.

```
# Following locationList and tagList are examples for PASTA
# to plot Shear Stress, N1 and N2 as a function of time.
```

```
locationList = [
'StepInfo.Time',
'StepData.TotalStress.Shear',
'StepData.TotalStress.N1',
'StepData.TotalStress.N2'
]
tagList = [
'Time',
'Shear Stress',
'N1',
'N2'
]
```

Description :

The `locationList` is the list of the variables in the Output UDF that should be added to the `GraphSheet`, and the `tagList` is the list of the tag names for those variables.

Usage :

(a) Read the Output UDF
Open the Output UDF by GOURMET.

(b) Load and execute the Python script
Load the script "`pasta_python.py`" into the Plot panel, and push the Run button. The data for gnuplot will be created and displayed in the `GraphSheet`.

(c) Check the `GraphSheet`
Click the `GraphSheet[]` in the Tree panel and check the `GraphSheet`.

(d) Execute Make
Push the Make button in the Plot panel. A script for gnuplot will be generated in the Plot panel. In addition, a text file "`plot.dat`" containing the same data as the `GraphSheet` will be created in the GOURMET starting directory.
The generated script for gnuplot will look like this:

```
# plot command template for platform
# datafile name is fixed as "plot.dat" in the current version
set title "GraphSheet[]"
plot "plot.dat" using 1:2 title 'Time' with lines , \
     "plot.dat" using 1:3 title 'Shear Stress' with lines , \
     "plot.dat" using 1:4 title 'N1' with lines , \
     "plot.dat" using 1:5 title 'N2' with lines
```

(e) Plotting the data
If you want plot the '`Shear stress`' as a function of '`Time`', modify the above script as follows, and push the Plot button.

```
# plot command template for platform
# datafile "plot.dat" is fixed current version
set title "GraphSheet[]"
plot "plot.dat" using 2:3 title 'Shear Stress vs. Time' with lines
#    "plot.dat" using 1:3 title 'Shear Stress' with lines , \
#    "plot.dat" using 1:4 title 'N1' with lines , \
#    "plot.dat" using 1:5 title 'N2' with lines
```

If you want to use plotting tools other than gnuplot, please read "`plot.dat`" into your favorite plotting applications.

An example of "`plot.dat`":
The first column is added automatically.

```
0   0.0     0.00237363   0.013161    -0.0120411
1   1.0     0.00331907   0.0125386   -0.0128968
2   2.0     0.00394328   0.0105911   -0.00743197
3   3.0     0.00478283   0.0098793   -0.00746133
4   4.0     0.00251843   0.0125421   -0.0110899
5   5.0     0.00236831   0.0152175   -0.0101838
...
```

Example of plot:

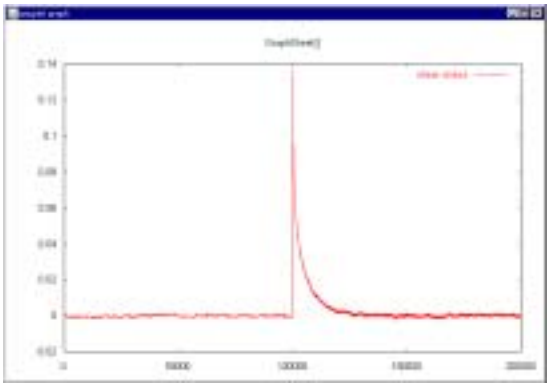

Figure 7.5: Example of a plot of stress relaxation.

2. "`pasta_plot_viscosity.py`"

This script is for plotting various viscosities depending on the value of `DeformationType` in the Output UDF.

```
# Plot Viscosity for shear, uniaxial, and biaxial deformation,
# and two Viscosities for planar deformation.
FlowType = $Simulation.Deformations[].FlowType
Strain = $Simulation.Deformations[].Strain
StrainRate = $Simulation.Deformations[].StrainRate
Deformation = $Simulation.Deformations[].DeformationType
nn = len(FlowType)
#
locationList = [
'StepInfo.Time',
'StepData.TotalStress.Shear',
'StepData.TotalStress.N1',
'StepData.TotalStress.N2'
]
shearList = [
'Time',
'Shear_Viscosity',
```

```
'N1/shear_rate^2',
'N2/shear_rate^2'
]
uniaxialList = [
'Time',
'Uniaxial_Viscosity',
'non-data1',
'non-data2',
]
biaxialList = [
'Time',
'Biaxial_Viscosity',
'non-data1',
'non-data2',
]
planarList = [
'Time',
'Planar_Viscosity_p1',
'Planar_Viscosity_p2',
'non-data',
]
```

Usage :
The usage is almost the same as "pasta_python.py".

Example plot:

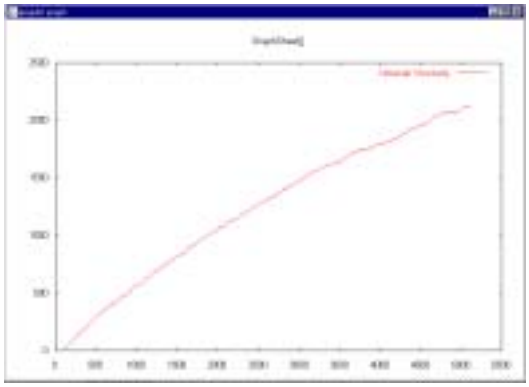

Figure 7.6: Example of a plot of uniaxial elongational viscosity.

3. "pasta_stress.py"

This script should be run from the Python panel of GOURMET. It creates a text file listing the stress components as function of time.

```
#for cygwin
filename = "E:\\somewhere\\stress.txt"
#for unix or linux
#filename = "/somewhere/stress.txt"
t = $StepInfo.Time
if t == None:
print "no data in this record"
```

```
else:
fout = open(filename,'a')
totalshear = $StepData.TotalStress.Shear
totalN1 = $StepData.TotalStress.N1
totalN2 = $StepData.TotalStress.N2
outstr = "%f\t%f\t%f\t%f\n" %(t,totalshear,totalN1,totalN2)
print outstr
fout.write(outstr)
```

Description :

The variable `filename` should be set to the full path name of the output file. You can modify the script for outputting the data other than the stress.

The usage :

(a) Read the output UDF
Open the Output UDF by GOURMET.

(b) Load the Python script
Load "`pasta_stress.py`" into the Python panel (not the Plot panel), and change the 'filename' to the name of the output file you want to create.

(c) Execute the script
Push the Run button.

(d) Check the output file.
Check the contents of the output file.

Example of the output file:

```
0.000000    -0.007746   -0.026151   0.017489
10.000000   -0.006312   -0.012012   0.009465
20.000000   -0.003527   -0.017801   0.018997
30.000000   -0.001595   -0.005478   0.013647
40.000000   -0.002701   -0.017810   0.019853
50.000000    0.006846   -0.028618   0.024955
60.000000    0.000780   -0.011298   0.009039
```

4. "`pasta_for_gt2gw.py`"

This script creates the input file for `gt2et` and `gt2gw` from an Output UDF of PASTA. `gt2et` and `gt2gw` are described in the next section. This script should also be run from the Python panel of GOURMET.

```
#for cygwin
filename = "k:\\somewhere\\gt.txt"
#for unix or linux
#filename = "/somewhere/gt.txt"
t = $StepInfo.Time
if t == None:
print "no data in this record"
else:
fout = open(filename,'a')
totalshear = $StepData.TotalStress.Shear
totalN1 = $StepData.TotalStress.N1
totalN2 = $StepData.TotalStress.N2
```

```
gamma = $StepInfo.Gamma
if gamma != 0:
outstr = "%f\n" %(totalshear/gamma)
fout.write(outstr)
else:
print "gamma == 0 "
```

Usage :

(a) Read the Output UDF
Open the Output UDF by GOURMET.

(b) Load the script
Load "pasta_for_gt2gw.py" into the Python panel, and change the 'filename' to the name of the output file you want to create.

(c) Execute the script
Push the Run button.

(d) Check the output file.
Check the contents of the output file.

Example of the output file:

```
0.287674
0.263989
0.253326
0.235426
0.226458
0.214798
0.203959
0.196313
0.190866
0.195164
0.183709
```

7.3 Rheology data analysis tools

Three programs are available for the data analysis. The input files for these programs are plain text files (not UDF files), and can be conveniently generated from the Output UDF by running "pasta_for_gt2gw.py" in GOURMET.

The rheology data analysis programs are in the following directory.

```
PF_ENGINE/PASTA/tools
```

The following three programs are available.

♢ gt2et : Calculate linear stress growth function $\eta^+(t)$ from relaxation modulus $G(t)$

♢ gt2gw : Calculate complex viscosity $\eta^*(\omega)$ or complex modulus $G^*(\omega)$ from relaxation modulus $G(t)$

♢ smooth : Perform smoothing of $G(t)$, $\eta^+(t)$ or $G^*(\omega)$

`gt2et`

Calculate the linear stress growth function $\eta^+(t)$ from the relaxation modulus $G(t)$

$$\eta^+(t) = \int_0^t G(t)dt \tag{7.1}$$

- Usage :

```
gt2et [-s dt] < infile > outfile
 -s dt   : specify the time step dt. The default of dt is 1.0.
```

- Input file format: Each line contains single data of $G(t)$, as follows:

```
G(t=0)
G(dt)
G(2dt)
...
```

- Output file format: t and $\eta^+(t)$ are output in a single line, with a tab as a separator.

```
0     η+(0)
dt    η+(dt)
2dt   η+(2dt)
...
```

`gt2gw`

Calculate the complex viscosity $\eta^*(\omega)$ or the complex modulus $G^*(\omega)$ from the relaxation modulus $G(t)$.

$$\eta^*(\omega) = \int_0^\infty G(t)e^{i\omega t}dt \tag{7.2}$$

$$G^*(\omega) = i\omega \int_0^\infty G(t)e^{i\omega t}dt \tag{7.3}$$

- Usage :

```
gt2gw [-n m] [-s dt] [-e] < infile > outfile
 -n m : The number of data used for FFT N = 2^m. The default is m=16
 -s dt : The time step dt. The default of dt is 1.0.
 -e : Output η*(ω). G*(ω) is output by default.
```

- Input file format: Same as the input file for `gt2et`.

```
G(t=0)
G(dt)
G(2dt)
...
G(Mdt)
```

  Note: You have to choose the `m` value satisfying $N = 2^m > M$. $N > 4M$ is desirable.

- Output file format: The angular frequency ω, real and imaginary parts of $G^*(\omega)$ are written in a single line. The interval of ω is $d\omega = 2\pi/(Ndt)$.

```
0        G′(0)          G″(0)
dω       G′(dω)         G″(dω)
2dω      G′(2dω)        G″(2dω)
..       ..             ..
N/2dω    G′(N/2dω)      G″(N/2dω)
```

`smooth`

Perform smoothing of $G(t)$, $\eta^+(t)$ or $G^*(\omega)$. The smoothed data are convenient for the logarithmic plot of $G(t)$ etc. The first or second order Savitzky-Golay filter is used.

- Usage :

```
smooth [-n ndiv] [-r r] [-x nmax] [-s skip] [-2] [file]
 -n ndiv   : Output ndiv points in one decade. The default is 20.
 -r r      : For smoothing, use the data points whose distance from the current
             data point is smaller than r times the distance to the previous data point.
             The default is 2.0.
 -x nmax   : At most nmax data points to the left and right of the current data point
             will be used for smoothing. The default is 1000.
 -s skip   : The first skip data are output without carrying out smoothing.
             The default is 0.
 -2        : Fit the data by a second order polynomial. The default is to use a first
             order polynomial.
 file      : Input file. The default is the standard input.
```

  The results are written to the standard output.

- Input file format: The measurement time t and measured value $v(t)$ in each line.

```
t0   v(t0)
t1   v(t1)
t2   v(t2)
..   ..
```

  The measurement times should be in increasing order $\mathtt{t}_0 < \mathtt{t}_1 < \mathtt{t}_2 < ...$, but they need not be equally spaced.

- Output file format:

```
T0   v̄(T0)
T1   v̄(T1)
T2   v̄(T2)
..   ..
```

  Output points $\mathtt{T}_0, \mathtt{T}_1, \mathtt{T}_2, ...$ are equally separated in the logarithmic scale. There are `ndiv` output points in one decade. `v̄(T)` is the smoothed result.

  If you want to apply smoothing to the output of `gt2gw`, which contains three columns in each line $(\omega, G^{'}, G^{''})$, it is convenient to use the script "`gwsmooth`":

```
gwsmooth infile > outfile
```

7.4 Animation program `pastanim`

As described in section 5.2.1, PASTA-2.8 (or later) can output a trajectory file which contains the data of the motion of a selected chain. `pastanim` can read the trajectory file and show an animation of the chain motion.

- Starting `pastanim`

  Enter the following line into the command prompt:

```
pastaim [-w window_size] trjectory_file
```

  Here, `window_size` (optional) is the window size in pixels (default 600), and `trajectory_file` is the name of the trajectory file you want to visualize.

- Using `pastanim`

  You can use the following mouse and keyboard commands:

  Mouse operation:

`drag`	rotate
`Shift+drag`	zoom in/out
`Ctrl+drag`	translate

  Keyboard command:

`Q` or `q`	quit pastanim
`Return`	start/stop animation
`Space`	go to next frame
`b`	go back to the previous frame
`h j k l`	rotate
`i o`	zooom in/out
`r`	go to the first frame
`R`	go to the first frame, and reset zoom and rotation
`e`	go to the last frame
`G`	go to any frame (enter the frame number from the console)
`c`	reset translation
`C`	reset translation, zoom, and rotation
`W`	write current frame into a ppm file

Appendix A

Compiling PASTA

A.1 How to compile PASTA and FORK

Compilation of PASTA and/or FORK from the source code is very simple. Go to the directory `PASTA/PASTA-2.7/src` or `PASTA/FORK-1.4.1/src`, and type 'make'. The `Makefile` in the directory will automatically figure out the operating system and CPU you are using.

If 'make' fails, please check if the environment variable `PF_FILES` is correctly set to the directory where GOURMET is installed. For example, header files such as "`pfinterface.h`" should be found in the directory `$PF_FILES/include/`.

If 'make' succeeds, please copy the new binary manually into the appropriate directory, such as `$PF_ENGINE/bin/cygwin/`.

References

1) Doi, M. and Edwards, S. F. eds.: *The Theory of Polymer Dynamics*, Oxford University Press (1986).

2) Takimoto, J., Tasaki, H. and Doi, M.: Predictions of the rheological properties of polymer melts by stchastic simulation, in *Proceeding of XIIIth International Congress on Rheology*, p. 97 (2000).