OCTA

Integrated simulation system for soft materials

SIMULATION UTILITIES FOR SOFT AND HARD INTERFACES

version 10.5

USER'S MANUAL

OCTA User's Group

OCT. 22 2017

Authors of the Manual

Chapter 1	Hiroshi Morita, Hiroya Kodama, Takashi Honda and Toshihiro Kawakatsu
Chapter 2	Hiroshi Morita, Hiroya Kodama, Takashi Honda and Toshihiro Kawakatsu
Chapter 3	Takashi Honda
Chapter 4	Katsuyuki Yokomizo, Hiroya Kodama and Takashi Honda
Chapter 5	Takashi Honda
Chapter 6	Takashi Honda
Chapter 7	Toshihiro Kawakatsu
Chapter 8	Shinji Urashita
Appendix B	Takashi Honda
Appendix C	Takashi Honda
Appendix D	Hiroya Kodama

Programers

SUSHI	Takashi Honda, Hiroya Kodama, Jiunn-Ren Roan, Hiroshi Morita and Toshihiro Kawakatsu
CPC: The Simple Python scripts for χ-parameter guess	Toshihiro Kawakatsu
SPCF: The tool for correlation function analysis	Shinji Urashita
InterfaceSimulator	Hiroya Kodama

Acknowledgement

This work was 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.

The implemantation of parallel computation of SUSHI9 for GPU and MPI has been supported by the Global Scientific Information and Computing center(GSIC) in Tokyo institute of technology with supercomputer TSUBAME2.0. We thank Prof. T. Aoki, Prof. T. Watanabe, Mr. J. Sasaki, and Mr. Y. Matsumoto for giving T. Honda special education for using TSUBAME2.0.

The implemantation of parallel computation of SUSHI9 for MPI also has been supported by the Information Initiative Center in Hokkaido university with super computer SR16000/M1. We thank Prof. M. Omiya in Hokkaido university, Dr. K. Hagita in national defense academy of Japan, and Dr. N. Goota and Dr. Y. Hirose in Hitachi, Ltd. for giving T. Honda many comments and advices.

We also thnak members of the technical seminar of polymer simulation organized by Japan Association for Chemical Innovation, for giving us many comments and opportunities of discussions.

The implemantation of parallel computation of SUSHI9 used computational resources of the K computer provided by the RIKEN Advanced Institute for Computational Science through the HPCI System Research project (Project ID:hp130056). We thank members of Research organization for Information Science & Technology (RIST), for giving T. Honda many comments and advices.

Contents

List of Figures

List of Tables

Chapter 1

What is SUSHI?

In Octa project, several simulators for polymeric materials based on mesoscale models are under development. The simulator explained in this manual is named "SUSHI" (Simulation Utilities for Soft and Hard Interfaces). The phenomena that can be dealt with by SUSHI are over the length scale of 10-100 nm and the time scale of $10^{-8} \sim 1$ sec. Typical structures characteristic to such length scales are the interfaces of polymer melts or polymer solutions. Therefore, SUSHI can be regarded as a simulator for the interfacial structures of polymer systems. One of the advantages of SUSHI is that it can simulate heterogeneous systems, for example micro-phase separations and micelles. Moreover, by analyzing the numerical data obtained by SUSHI, one can obtain physical quantities relevant to the interfacial tension, critical micelle concentration, and so on.

SUSHI can carry out static and dynamic simulations based on Self Consistent Field (SCF) theory [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]. Hereafter, the dynamic calculation will be called "dynamic mean field method" (or "dynamic density functional method"). There are some other techniques that can treat the phenomena on the intermediate length and time scales. The "Phenomenological Density Functional(PDF)" based on the functional Taylor series expansion of the free energy of the system is one such example [11, 12] .

Although the calculation cost of the SCF method is higher than the PDF method, the SCF method can correctly take the conformation of the chains into consideration (within the resolution of the coarse-grained length scale).

As one can confirm in table 1.1, SUSHI is designed as flexible as possible. Any chain topology and monomer (segment) sequence can be dealt with. Effective bond length and specific volume of segments can be arbitrarily specified. According to the problem, one can select a suitable spatial mesh, such as regular, rectangular, spherical, or cylindrical mesh. Boundary conditions can also be selected as periodic boundary, solid wall, or reflective wall. Both canonical and grand canonical ensembles are supported. Functions to graft polymers onto a wall or to confine polymers in a certain region are also available. Owing to these flexibility and generality of SUSHI, it can be applied to a wide variety of problems, and SUSHI is expected to be very useful in various scenes in the material design.

In order to guarantee such high flexibility, various extensions of the theory are made in SUSHI, and therefore the basic mathematical formalism is a little complicated. In order to avoid complexity, in Chapter 2, we start describing the outline of the basic concepts and formalism of the SCF theory using a homopolymer blend system as a simple example. In Section 2.9, we describe the details of the numerical techniques to solve the theoretical equations. Chapter 3 is devoted to a simple guide for operating SUSHI. Chapter 4 is for some introductory exercises for using SUSHI, where the following problems are discussed; The interfacial physical properties, such as the toughness of the interfaces in polydisperse polymer blends, the interaction between surfaces onto which polymers are adsorbed, and the critical micelle concentration of block copolymers in a solvent (i.e. surfactant systems), and so on. The details of the input parameters for SUSHI are explained in Chapter 5.

Usage of simple Python tools to draw graphs or pictures from the numerical data of SUSHI on GOURMET is described in Chapter 6.

Although the Flory's χ-parameters are used in SUSHI as important input parameters to specify the character of the system, the procedure to estimate these parameters is non-trivial. To eliminate such a difficulty, in Chapter 7, we offer Python scripts named "CPC" (Chi Parameter Calculator) that can be used to estimate these χ-parameters.

In order to calculate the real space correlation functions of the domain morphology obtained by the

simulation, a simulator "SPCF" (Spatial Correlation Function calculator) is prepared. In Chapter 8, we describe how to use this SPCF and show several examples.

How to compile the source codes and how to extend the simulator are briefly described in the appendices. There is a special-purpose simulator called "InterfaceSimulator" that uses the basic functions of SUSHI. The outline of this simulator is described in the appendices.

Fixed bugs by SUSHI10.0

- The conformational entropy effect of a polymer was lacked in the free energy calculation when the canonical ensamble and any wall boundary conditions were satisfied.
- Powder pattern of scattering function was not circular averaged.

Fixed bugs by SUSHI10.3

- Strong polyelectrolite calculations could not be converged.
- Outputs of cylindrical mesh calculations were abnormal with asymmetric mesh sizes.
- Cylindrical mesh calculations with walls (at cylinder ends) could not be converged.
- Action plot_1D outputs were abnormal with walls.
- SUSHIInput.zoom.sigma_per_b was miss type and SUSHIInput.zoom.b_per_sigma was added. The SUSHIInput.zoom.sigma_per_b in UDF of previous versions is recognized as SUSHIInput.zoom.b_per_sigma.
- Zooming options with -Z did not output coordinates of beads.
- Soft particles calculations had a bug in periodic BDC treatment.
- Calculations including solvents + wall had a bug.

Fixed bugs by SUSHI10.4

- Abnormal terminations with zero values of constV and constW.
- Abnormal terminations of MPI+GPU+hydrodynamics calculations.

Table 1.1: Functions of SUSHI 1/2

	Item	Function	
1	Purpose	Mesoscale simulator based on the dynamical mean field theory for polymer melts and polymer solutions	
2	Version	Version 9.2	
3	Programming language	C++	
4	Compilers	Microsoft VC++ Ver. 2010 or higher g++ library for gcc Ver. 4 or higher	
5	Mean field method	Self consistent field method with path integral formalism Code name: SCF (Self Consistent Field)	Ginzburg-Landau theory using Random Phase Approximation Code name: GRPA (GL using RPA)
6	Polymer architecture Topology Sequence Character of segment	SCF Any type Any type Effective bond length, Specific volume	GRPA Any type Any type Effective bond length
7	System State Boundary condition Mesh	 Polymer melt Polymer solution(including void) Periodic, Reflective, Wall Regular mesh 1D∼3D Rectangular mesh (Cartesian mesh with inhomogeneous mesh width) 1D∼3D Cylindrical mesh 2D Spherical mesh 1D	
8	Calculation method	Static equilibrium (Eq), Dynamics (Dy), Monte Carlo (Eq)	
9	Ensemble	Canonical ensemble Grand canonical ensemble (SCF)	
10	External field	Surface force due to wall(SCF)	
11	Grafting	Grafting polymer to wall(SCF)	
12	Scheme for calculating path integral	Explicit scheme, Implicit scheme (SCF)	
13	Scheme for dynamic equation (Dy)	Explicit scheme, Implicit scheme	
14	Mask of free end of polymer (Eq)	Micelle can be simulated by restricting the free ends of polymers to within a certain region. (SCF)	
15	Common use of path integral (Eq)	Computation time is reduced by commonly using the same path integral for similar polymer structures. (SCF)	
16	Domain specification (Eq)	Target morphology is realized by giving appropriate initial conditions for the self-consistent fields. (SCF)	
17	Radius of gyration	The radius of gyration of a sub-chain and full chain can be calculated. (SCF)	
18	Mobility (Dy)	Value and type of mobility of segment can be specified.	
19	Adsorption dynamics (Dy)	The adsorption dynamics onto wall is available. (SCF)	
20	Chemical reaction dynamics (Dy)	1)Rapid reaction 2)Active-site reaction (SCF) 3)Grafting reaction (SCF)	
21	Polyelectrolyte	The strong polyelectrolyte simulation is available.	
22	Quench	χ parameter can be changed in the dynamic simulation. (Dy)	
23	Sheared dynamics (Dy)	Shear can be introduced to dynamics. (SCF)	
24	Compressible dynamics (Dy)	Compressibility can be introduced to dynamics. (SCF)	
25	Thermal fluctuation (Dy)	Thermal fluctuation can be introduced to dynamics. (SCF)	
26	System size optimization	System size can be optimized to minimize the free energy density.	
27	Scattering function	The Scattering function between segments can be calculated.	

"Eq" means that the function is valid in the static equilibrium calculation.
"Dy" means that the function is valid in the dynamic calculation.

Table 1.2: Functions of SUSHI 2/2

	Item	Function
28	GRPA (Dy)	Dynamic density functional theory based on Ginzburg-Landau (GL) theory using Random Phase Approximation (RPA)
29	Hybrid (Dy)	Density functinoal theory hybridizing SCF and GRPA
30	Hydrodynamics (Dy)	Dynamic density functional theory introducing hydrodynamic effects
31	External Electric Field (Dy)	Dynamic density functional theory introducing un external electric field
32	Obstacles (Eq)	1)Particles can exist in systems as obstacles in SCF calculations. The region of an obstacle can be selected as either inner side or outer side of the particle's shell(SCF). 2)Fibers can exist in systems as obstacles in SCF calculations. The region of an obstacle can be selected as either inner side or outer side of the particle's shell(SCF). 3)Surface chi parameters χ_s can be set on obstacles. 4)Ends of polymers can be grafted on obstacles.
33	Zooming	SUSHI supports to make the input file for COGNAC.
34	Modification of the parallel calculation	Wall can be introduced (MPI,GPU) but only for static calculation. MPI calculation can accept no divided axes.
35	The obstacle defined by triangle polygon data	Any shape of obstacles can be inputted, which is defined by triangle polygons.
36	Soft particles	The particle defined by fixed shape of solvent can be inputted.
37	Hydrodynamic effects to MPI + GPU	Hydrodynamic effects can be introduced to MPI+GPU calculations.
38	Bugs fix	Unexecutable bugs are fixed with MPI + GPU.

"Eq" means that the function is valid in the static equilibrium calculation.
"Dy" means that the function is valid in the dynamic calculation.

Chapter 2

Theoretical background

2.1 Self consistent field theory

In this chapter, we describe the outline of the Self-Consistent Field (SCF) theory using linear homopolymer chains and their blends as examples.

We consider a system with volume $\mathcal{V}$. Boundary conditions are decided according to the physical properties of the system at the boundaries, such as periodic boundary conditions, Neumann boundary conditions, and Dirichlet boundary conditions. The minimum unit that constitutes the system is called a segment, and the polymers are composed of several kinds of segments specified by the index $K = 1, 2, \cdots$. The polymer species are specified by the index $p = 1, 2, \cdots$. The degree of polymerization of the p-type polymer chain is denoted as $N^{(p)}$, and the total number of p-type polymer chains in the system is M_p. We number the segments on the p-type chain with the index $i = 0, 1, 2, \cdots, N^{(p)}$ from one end of the polymer. Note that, in homopolymer blend systems, the index K for the segment species can be identified with the index p for the polymer species. As we use only homopolymer blends in this chapter, the index K will be used instead of the index p.

In the SCF theory, we can calculate the equilibrium concentration (or the volume fraction) distribution $\phi_K(\mathbf{r})$ of K-type segments taking the chain conformations into account using the path integral formalism. To give a simple explanation of the path integral formalism of polymers, we adopt a random walk model on a cubic lattice with lattice constant b. In the SCF theory, the chain conformations are treated statistically using probability distributions of conformations. To calculate the statistical distribution of the chain conformation, we focus on a single chain in a system composed of mutually interacting many chains, and approximate this tagged chain by a single ideal chain in a mean field. For the reason mentioned below, this mean field is called self consistent field. The self consistent field is a potential field that accounts for the external conditions such as the interaction between segments and the incompressible conditions, and has the following form

$$V_K(\mathbf{r}) = W_K(\mathbf{r}) + \Big[\text{potential decided by incompressible condition}\Big], \tag{2.1}$$

where $W_K(\mathbf{r})$ is the mean field resulting from the interaction between segments, and is given by

$$W_K(\mathbf{r}) = \sum_{K'} \chi_{KK'} \phi_{K'}(\mathbf{r}). \tag{2.2}$$

The 2^{nd} term on the right-hand-side of eq. (2.1) is the constraint force (it is equivalent to a Lagrange multiplier) due to the constraints imposed on the system, such as the incompressible condition. In Section 2.5, we give the definite forms of the constraint force for some constraining conditions.

Under such mean field approximation using the self consistent field, we can divide a polymer chain into subchains whose conformations are statistically independent of each other. Therefore, the calculation of the statistical probability distribution of the conformation of the whole chain can be reduced to the calculation of those of individual subchains. In the following, we show how to calculate such statistical probability distributions of conformations of subchains using path integrals.

We consider a homopolymer chain composed of K-type segments, and focus on a section of the chain (subchain) between the segment i and the segment j. Let us define a quantity $Q_K(i, \mathbf{r}_i; j, \mathbf{r}_j)$ (so-called path integral) as the equilibrium statistical weight of this subchain to have a conformation with the i-th and the

j-th segments at $\mathbf{r}_i$ and $\mathbf{r}_j$. If we write the self consistent field (mean field) acting on the K-type segment at position $\mathbf{r}$ as $V_K(\mathbf{r})$, the statistical weight of the subchain with its segments locating at $\mathbf{r}_i, \mathbf{r}_{i+1}, \cdots, \mathbf{r}_{j-1}, \mathbf{r}_j$ is expressed as

$$\exp\Big[-\beta\Big\{\frac{1}{2}V_K(\mathbf{r}_i) + V_K(\mathbf{r}_{i+1}) + \cdots + V_K(\mathbf{r}_{j-1}) + \frac{1}{2}V_K(\mathbf{r}_j)\Big\}\Big] \tag{2.3}$$

apart from a constant factor. Here, $\beta = 1/k_{\mathrm{B}}T$ and the contributions from the end segments are assumed to be one half of those from the inner segments because a junction point is shared by two subchains meeting at that segment. By using eq. (2.3), the path integral $Q_K(i, \mathbf{r}_i; j, \mathbf{r}_j)$ is given by a sum of Boltzmann factors for all possible conformations under the conditions that the both ends are fixed at positions $\mathbf{r}_i$ and $\mathbf{r}_j$, respectively. Therefore, we have

$$Q_K(i, \mathbf{r}_i; j, \mathbf{r}_j) = \frac{1}{z^{|i-j|}} \sum_{\text{all conformation}} \exp\Big[-\beta\Big\{\frac{1}{2}V_K(\mathbf{r}_i) + \sum_{k=i+1}^{j-1} V_K(\mathbf{r}_k) + \frac{1}{2}V_K(\mathbf{r}_j)\Big\}\Big], \tag{2.4}$$

where z is the number of nearest neighbor lattice sites. From eq. (2.4), the following recurrence formula can easily be derived.

$$Q_K(i, \mathbf{r}_i; j+1, \mathbf{r}) = \sum_{\mathbf{r}'} Q_K(i, \mathbf{r}_i; j, \mathbf{r}') \times \frac{1}{z} \exp\Big[-\frac{1}{2}\beta\big\{V_K(\mathbf{r}') + V_K(\mathbf{r})\big\}\Big] \tag{2.5}$$

where $\sum_{\mathbf{r}'}$ means the sum over the nearest neighbor lattice sites to $\mathbf{r}$. If we assume that Q_K varies slowly on the length scale of the lattice spacing, $Q_K(i, \mathbf{r}_i; j+1, \mathbf{r})$ can be expanded around $\mathbf{r} = \mathbf{r}'$, and we obtain

$$\begin{aligned} Q_K(i, \mathbf{r}_i; j+1, \mathbf{r}) \;=\; & \frac{1}{z} e^{-\beta V_K(\mathbf{r})/2} \sum_{\mathbf{r}'} \Big[e^{-\beta V_K(\mathbf{r})/2} Q_K(i, \mathbf{r}_i; j, \mathbf{r}) \\ & + \nabla\Big\{ e^{-\beta V_K(\mathbf{r})/2} Q_K(i, \mathbf{r}_i; j, \mathbf{r}) \Big\} \cdot (\mathbf{r} - \mathbf{r}') \\ & + \frac{1}{2}\nabla^2\Big\{ e^{-\beta V_K(\mathbf{r})/2} Q_K(i, \mathbf{r}_i; j, \mathbf{r}) \Big\} : (\mathbf{r} - \mathbf{r}')(\mathbf{r} - \mathbf{r}') + \cdots \Big]. \end{aligned} \tag{2.6}$$

We further assume that $\beta V_K(\mathbf{r})$ are small, then eq. (2.6) reduces to the following Schrödinger type time evolution equation

$$\frac{\partial}{\partial i} Q_K(i', \mathbf{r}'; i, \mathbf{r}) = \Big[\frac{b^2}{6}\nabla^2 - \beta V_K(\mathbf{r})\Big] Q_K(i', \mathbf{r}'; i, \mathbf{r}). \tag{2.7}$$

Here, the lattice constant b corresponds to the average distance between a segment and its adjacent segment along the chain, and is called the effective bond length, which in general depends on the segment species K. When we regard the segment index i as time, the evolution equation eq. (2.7) is identified with the diffusion equation. Thus, the chain conformation can be regarded as a path of a Brownian particle in the self consistent field. The initial condition for the partial differential equation eq. (2.7) is given by

$$Q_K(i', \mathbf{r}; i', \mathbf{r}') = \delta(\mathbf{r} - \mathbf{r}'), \tag{2.8}$$

which can easily be understood from the definition of the path integral.

Using the path integral obtained by solving eqs. (2.7) and (2.8), the concentration of the K-type segments at position $\mathbf{r}$ is expressed as

$$\phi_K(\mathbf{r}) = C_K \sum_i \int d\mathbf{r}_0 \int d\mathbf{r}_{N_K} Q_K(0, \mathbf{r}_0; i, \mathbf{r}) Q_K(i, \mathbf{r}; N_K, \mathbf{r}_{N_K}), \tag{2.9}$$

where N_K is the total number of segments composing the K-type polymer, C_K is the normalization constant, and $\mathbf{r}_0$ and $\mathbf{r}_{N_K}$ are the positions of the segments 0 and N_K at both ends of the chain, and $\sum_i$ means the sum over all N_K segments.

The expression of the normalization constant C_K in eq. (2.9) changes depending on whether the statistical ensemble of this chain is the canonical ensemble where the total number of chains in the system is fixed or the grand canonical ensemble where the system is in equilibrium with a reservoir of particles with a fixed concentration.

1) Canonical ensemble

$$C_K = \frac{M_K}{\int d\mathbf{r}_0 \int d\mathbf{r}_{N_K} Q_K(0, \mathbf{r}_0; N_K, \mathbf{r}_{N_K})}, \tag{2.10}$$

where M_K is the total number of chains in the system.

2) Grand canonical ensemble
We assume that the system is in chemical equilibrium with the external uniform reservoir (bulk system). Let us denote the concentration of the chains in the bulk as $\phi_K^{(\mathrm{bulk})}$. When $\phi_K(\mathbf{r}) \equiv \phi_K^{(\mathrm{bulk})}$, C_K is expressed in terms of the potential $W_K(\mathbf{r}) \equiv W_K^{(\mathrm{bulk})}$ calculated using eq. (2.2) as follows;

$$C_K = \frac{\phi_K^{(\mathrm{bulk})}}{N_K} \exp\Big[N_K(W_K^{(\mathrm{bulk})} + \mathrm{constant})\Big]. \tag{2.11}$$

The constant term on the right-hand-side originates from the arbitrariness in the self consistent field defined by eq. (2.1), and we drop this term without loss of generality. Such a procedure corresponds to assuming no constraints in the bulk phase in the grand canonical ensemble.

For the actual calculations, it is convenient to integrate $Q_K(i, \mathbf{r}; N_K, \mathbf{r}_{N_K})$ and $Q_K(0, \mathbf{r}_0; i, \mathbf{r})$ (path integrals) appearing in the formula eq. (2.7) with respect to their initial positions, and introduce $Q_K(i, \mathbf{r}_i)$ and $\widetilde{Q}_K(i, \mathbf{r}_i)$ as follows.

$$Q_K(i, \mathbf{r}_i) = \int d\mathbf{r}_0 Q_K(0, \mathbf{r}_0; i, \mathbf{r}_i) \tag{2.12}$$

$$\widetilde{Q}_K(N_K - i, \mathbf{r}_i) = \int d\mathbf{r}_{N_K} Q_K(N_K, \mathbf{r}_{N_K}; i, \mathbf{r}_i). \tag{2.13}$$

Then, the path integrals $Q_K(i, \mathbf{r}_i)$ and $\widetilde{Q}_K(i, \mathbf{r}_i)$ are obtained by solving the evolution equations

$$\frac{\partial}{\partial i} Q_K(i, \mathbf{r}) = \Big[\frac{b^2}{6}\nabla^2 - \beta V_K(\mathbf{r})\Big] Q_K(i, \mathbf{r}) \tag{2.14}$$

$$\frac{\partial}{\partial i} \widetilde{Q}_K(i, \mathbf{r}) = \Big[\frac{b^2}{6}\nabla^2 - \beta V_K(\mathbf{r})\Big] \widetilde{Q}_K(i, \mathbf{r}) \tag{2.15}$$

using the following initial conditions.

$$Q_K(0, \mathbf{r}) = \widetilde{Q}_K(0, \mathbf{r}) = 1. \tag{2.16}$$

By using the path integrals obtained from eqs. (2.14) and (2.15), the segment density at position $\mathbf{r}$ is given using eqs. (2.9), (2.12) and (2.13) as follows.

$$\phi_K(\mathbf{r}) = C_K \sum_i Q_K(i, \mathbf{r}) \widetilde{Q}_K(N_K - i, \mathbf{r}). \tag{2.17}$$

As is shown in eq. (2.1), the self consistent field $V_K(\mathbf{r})$ depends on the segment concentration $\phi_K(\mathbf{r})$. Then, $\phi_K(\mathbf{r})$ is calculated from the path integral $Q_K(i, \mathbf{r})$ through eq. (2.17). Finally, the path integral $Q_K(i, \mathbf{r})$ is obtained by solving eqs. (2.14) and (2.15) that contain the self consistent field $V_K(\mathbf{r})$. Therefore, $V_K(\mathbf{r})$, $\phi_K(\mathbf{r})$ and $Q_K(i, \mathbf{r})$ define themselves in a recursive manner as shown in fig. 2.1. This diagram shows that the three quantities should be obtained in a self consistent manner. This is the origin of the name of the self consistent field.

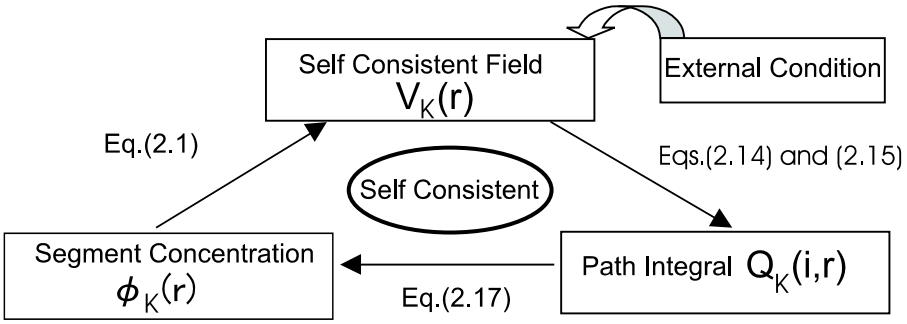


Figure 2.1: The basic scheme of the self consistent field (SCF) theory

2.2 Extension to branched chains and block copolymers

When we treat block copolymers, branched polymers, etc. by SCF, an extension of the treatment of the homopolymer blends described above is necessary. For such a case, as is shown in fig. 2.2, we have to divide the polymer into a set of linear subchains. Here we introduce an index r to specify the subchains. The connecting points of subchains are called junctions. Let us assume that the r-th subchain of the p-th polymer consists of $N_r^{(p)}$ segments. Then, these $N_r^{(p)}$ segments are numbered from its one end to the other using an index i.

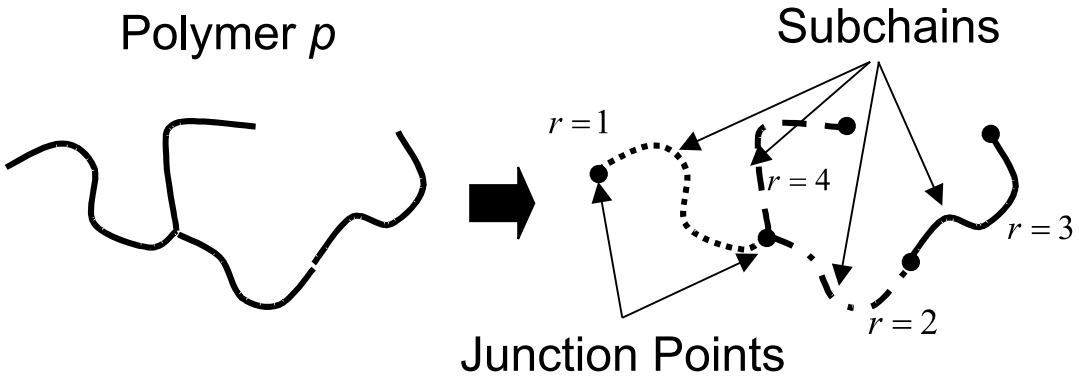


Figure 2.2: Dividing a chain into subchains

The same statistical treatment on the chain conformation as that adopted in the preceding section for the linear polymers can also be used for the present case. When we divide a branching polymer chain into some subchains (fig. 2.2), these subchains are statistically independent under the mean field approximation. Therefore, the conformational statistics of the whole polymer chain is obtained by independently calculating the statistics of each subchain. The method to calculate the conformational statistics of the subchains using path integrals is almost the same as that for the linear homopolymers but with only minor modifications. In the following, the technique for calculating the path integrals for branched chains is described.

In the linear homopolymer case discussed in Section 2.1, both ends of a polymer chain were free ends. On the other hand, in the case of branched chains or block copolymers, in general, some other subchains are connected to the end segments of the subchain. Therefore, when calculating the statistical weight of the conformation of a subchain, one has to take into consideration the statistical weight of all the subchains connected to the end segments of the target subchain. In this case, the contribution from the r-th subchain of the p-th polymer to the segment concentration at position $\mathbf{r}$ is expressed as

$$\phi_r^{(p)}(\mathbf{r}) = C_r^{(p)} \sum_i \int d\mathbf{r}_0 \int d\mathbf{r}_N q_0(\mathbf{r}_0) Q_K(0, \mathbf{r}_0; i, \mathbf{r}) Q_K(i, \mathbf{r}; N, \mathbf{r}_N) q_N(\mathbf{r}_N), \tag{2.18}$$

where $Q_K(i, \mathbf{r}_i; j, \mathbf{r}_j)$ is the same path integral calculated using eqs. (2.7) and (2.8), and K is the segment species (namely, $K_r^{(p)}$) that constitutes this subchain. $C_r^{(p)}$ is the normalization constant, N is the total number of segments contained in this subchain (namely $N_r^{(p)}$), $\mathbf{r}_0$ and $\mathbf{r}_N$ are the positions of the segments 0 and N at both ends, and $q_0(\mathbf{r}_0)$ and $q_N(\mathbf{r}_N)$ express the statistical weights of the remaining parts of the whole chain connected to each end segment. The sum $\sum_{i=0}^{N}$ should be taken over all $N+1$ segments belonging to the target subchain. For the physical meaning of eq. (2.18), readers should refer fig. 2.3.

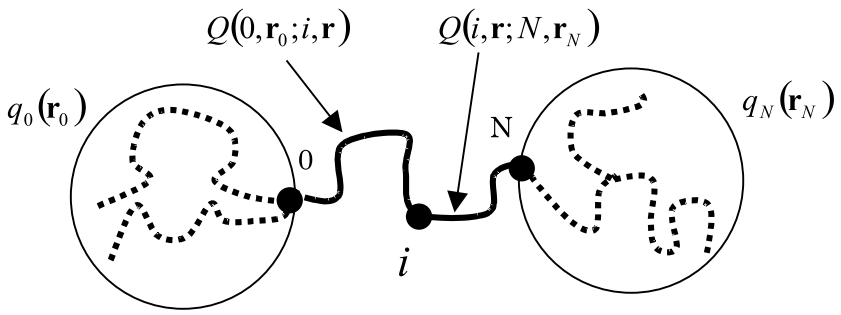


Figure 2.3: Schematic picture showing the physical meaning of eq. (2.18)

Similarly to eqs. (2.10) and (2.11), the expression of the normalization constant in eq. (2.18) depends on whether the statistical ensemble of the polymer to which the target subchain belongs is the canonical ensemble or the grand canonical ensemble.

1) Canonical ensemble

$$C_r^{(p)} = \frac{M_r^{(p)}}{\int d\mathbf{r}_0 \int d\mathbf{r}_N q_0(\mathbf{r}_0) Q_K(0,\mathbf{r}_0;N,\mathbf{r}_N) q_N(\mathbf{r}_N)}, \tag{2.19}$$

where $M_r^{(p)}$ is the total number of the subchains of type (p,r) contained in the system. It is obviously equal to the total number of p-th chains. Similarly, we can easily show that $C_r^{(p)}$ does not depend on the index r of the subchain but on the type of the chain p.(Therefore, we can denote this as $C^{(p)}$.)

2) Grand canonical ensemble

$$C_r^{(p)} = \frac{\phi_p^{(\mathrm{bulk})}}{\sum_{r''} N_{r''}^{(p)}} \exp\Big[\sum_{r'} N_{r'}^{(p)} W_{K_{r'}^{(p)}}^{(\mathrm{bulk})}\Big], \tag{2.20}$$

where $\phi_p^{(\mathrm{bulk})}$ is the equilibrium bulk concentration of the p-type polymer. Similarly to the canonical case, we can show that $C_r^{(p)}$ does not depend on the subchain type r but only on the chain type p.(Thus, we can write $C^{(p)}$.)

Similarly to eqs. (2.12) and (2.13) in the previous section, it is again convenient to use the following path integrals that are integrated over their initial positions,

$$Q_K(i,\mathbf{r}_i) = \int d\mathbf{r}_0 q_0(\mathbf{r}_0) Q_K(0,\mathbf{r}_0;i,\mathbf{r}_i) \tag{2.21}$$

$$\widetilde{Q}_K(N-i,\mathbf{r}_i) = \int d\mathbf{r}_N q_N(\mathbf{r}_N) Q_K(N,\mathbf{r}_N;i,\mathbf{r}_i). \tag{2.22}$$

These quantities obey the same evolution equations as eqs. (2.14) and (2.15) but with modified initial conditions given by

$$Q_K(0,\mathbf{r}) = q_0(\mathbf{r}) \tag{2.23}$$

$$\widetilde{Q}_K(0,\mathbf{r}) = q_N(\mathbf{r}). \tag{2.24}$$

Starting from these initial conditions, the path integrals are calculated by solving the evolution equations eqs. (2.14) and (2.15). Then, the contribution from this subchain to the segment density at position $\mathbf{r}$ is given by

$$\phi_r^{(p)}(\mathbf{r}) = C_r^{(p)} \sum_i Q_K(i,\mathbf{r}) \widetilde{Q}_K(N_r^{(p)} - i, \mathbf{r}). \tag{2.25}$$

The path integral for the whole chain can be obtained by performing the above-mentioned calculation on all the subchains. Then, the segment concentration $\phi_K(\mathbf{r})$ at position $\mathbf{r}$ is calculated by adding up eq. (2.25) for all subchains that consist of K-type segments.

$$\phi_K(\mathbf{r}) = \sum_p \sum_{r \in \text{the subchain of K species}} \phi_r^{(p)}(\mathbf{r}). \tag{2.26}$$

The definition of the self consistent field for branched chains is the same as that given by eq. (2.1), and should be determined in a self consistent manner as in Section 2.1.

2.3 Extension to systems with internal states

The formulation given in Sections 2.1 and 2.2 for subchains composed of a single kind of monomers can easily be extended to that for subchains composed of two or more kinds of monomers, for example

1) Tapered copolymers

2) Multi-state polymers where each segment can occupy either a ground state or one of the excited states

Calculation of the conformational statistics of these polymers becomes available by introducing the concept of the multi-states (two or more internal states) into the definition of the path integral given in Section 2.1.

A typical example is the tapered block copolymer that consists of two kinds of segments, *i.e.* A segment and B segment. In a tapered block copolymer, the A-segments and the B-segments are arranged randomly with changing their average compositions from one end of the chain to the other. In order to calculate the conformation of a tapered block copolymer, one has to calculate two path integrals $Q_A(i,\mathbf{r})$ and $Q_B(i,\mathbf{r})$ simultaneously. In solving such a coupled path integrals, the characteristic properties of the distributions of the segments along the chain (or subchain) are introduced through the following transition state probability matrix. (This is the matrix element for the calculation of the path integral in the direction from $i=0$ to $i=N$, where 0 and N are indices of the both ends of the chain.)

$$T_{KK'}(i) \equiv \left[\begin{array}{l} \text{The probability that (i + 1)th segment is K}' \text{ species,} \\ \text{when ith segment is K species.} \end{array} \right]. \tag{2.27}$$

Now we give an example of this transition state probability matrix using the A-B tapered block copolymer discussed above. Let us denote the average fractions of the A and B segments at i-th monomer as $f_A(i) = f(i)$ and $f_B(i) = 1 - f(i)$, respectively. If we assume that there is no correlation between the probability distributions at i th segment and the $i+1$ th segment, the elements $T_{KK'}(i)$ is given by

$$T_{AA}(i) = T_{BA}(i) = f(i+1), \quad T_{AB}(i) = T_{BB}(i) = 1 - f(i+1)\ . \tag{2.28}$$

By using the transition state probability matrix, the evolution equation of the path integral, eq. (2.14), is extended as follows:

$$\frac{\partial}{\partial i} Q_K(i,\mathbf{r}) = \sum_{K'} T_{KK'}(i) \Big[\frac{b_{K'}^2}{6} \nabla^2 - \beta V_{K'}(\mathbf{r}) \Big] Q_{K'}(i,\mathbf{r}), \tag{2.29}$$

where the dependence of the effective bond length b on the segment type K is explicitly described.

2.4 Free energy

The free energy of the system is composed of two contributions; one is the entropic contribution (mainly degrees of freedom of the chain conformation) and the other is the interaction between segments and the interaction between segments and the external fields. Using the path integrals introduced in the preceding sections, one can calculate the free energy taking the conformational entropy into consideration. As the branching structure and the effects of the conformational change can be taken into account in the SCF method, it is very useful in evaluating the free energy of the system composed of arbitrary chain architectures with sufficient accuracy. Such effects are almost neglected in the usual Flory-Huggins theory.

Here we skip all the details of the derivation of the expression of the free energy, and only show the final result. The Helmholtz free energy can be given in terms of the path integral as follows.

$$\begin{aligned} \mathcal{F}\big[\{\phi_K\},\{V_K\}\big] &= -k_{\mathrm{B}}T \sum_p M_p \ln \mathcal{Z}_p + \mathcal{W}\big[\{\phi_K\}\big] \\ &\quad - \sum_K \int d\mathbf{r} V_K(\mathbf{r}) \phi_K(\mathbf{r}) \\ &\quad + k_{\mathrm{B}}T \sum_p M_p \ln M_p. \end{aligned} \tag{2.30}$$

In the first term on the right-hand-side of eq. (2.30), $\mathcal{Z}_p$ is the partition function of a single chain of type p in the self consistent field, and is defined by

$$\mathcal{Z}_p = \sum_{\text{all conformation of chain}} \exp\Big[-\beta \sum_{\text{all segments i}} V_{K(i)}(\mathbf{r}_i)\Big], \tag{2.31}$$

where $K(i)$ is the species of the i-th segment. In case there are no loops in the chain, eq. (2.31) can further be written using the path integral as

$$\mathcal{Z}_p \;=\; \int d\mathbf{r} Q(i,\mathbf{r})\widetilde{Q}(N_r^{(p)}-i,\mathbf{r}), \tag{2.32}$$

where, the integration is taken over the position of an arbitrary chosen segment i on the subchain r.

As is easily understood from eq. (2.31), the first term on the right-hand-side of eq. (2.30) corresponds to the conformational entropy of the chain. The third term and the forth term are the contribution from the constraint conditions and that from the mixing entropy, respectively, the latter being a constant depending on the composition of the system. The second term on the right-hand-side of eq. (2.30) is the interaction between segments, whose explicit expression is given as follows.

$$\mathcal{W}\big[\{\phi_K\}\big] = \frac{1}{2}\sum_K \sum_{K'} \int d\mathbf{r} \epsilon_{KK'}\phi_K(\mathbf{r})\phi_{K'}(\mathbf{r}). \tag{2.33}$$

Here, $\epsilon_{KK'}$ is the interaction energy between a pair of segments K and K', and is related to the Flory's χ parameter by the following relation.

$$\chi_{KK'} = z\beta\big[\epsilon_{KK'} - \frac{1}{2}(\epsilon_{KK}+\epsilon_{K'K'})\big]. \tag{2.34}$$

Another way of evaluating the free energy is the Approximate Density Functional method, where the free energy is expressed as an approximate functional of the volume fraction ϕ_K. For example, as the free energy model for an AB homopolymer blend, one can use the Flory-Huggins free energy defined by

$$\frac{\mathcal{F}}{k_{\mathrm{B}}T} = \frac{\phi_A}{N_A}\ln\phi_A + \frac{1-\phi_A}{N_B}\ln(1-\phi_A) + \chi\phi_A(1-\phi_A). \tag{2.35}$$

or its power series expansion with respect to ϕ_A around its average value. The contribution from the spatial variation of the segment density ϕ_A is taken into consideration in the following Flory-Huggins-de Gennes free energy

$$\frac{\mathcal{F}\big[\{\phi_A(\mathbf{r})\}\big]}{k_{\mathrm{B}}T} = \int d\mathbf{r}\Big[\frac{\phi_A}{N_A}\ln\phi_A + \frac{1-\phi_A}{N_B}\ln(1-\phi_A) + \chi\phi_A(1-\phi_A) + \frac{b^2}{36\phi_A(1-\phi_A)}|\nabla\phi_A|^2, \tag{2.36}$$

where the relation $\phi_B = 1-\phi_A$ due to the incompressibility condition is used.

2.5 Chemical potential and constraint force

Suppose that the segment concentration is constrained to a given profile $\{\phi_K(\mathbf{r})\}$. Even if this given profile $\{\phi_K(\mathbf{r})\}$ is not an equilibrium profile, one can imagine an equilibrium state of chains under such a constraint on the segment concentration profile. In such a case, using the definition of the path integral and the free energy expression eq. (2.30), the chemical potential $\mu_K(\mathbf{r}) \equiv \delta\mathcal{F}/\delta\phi_K(\mathbf{r})$ acting on a K-type segment $(K = A, B, C, \cdots)$ at position $\mathbf{r}$ is expressed as

$$\mu_K(\mathbf{r}) = -V_K(\mathbf{r}) + \sum_{K'}\epsilon_{KK'}\phi_{K'}(\mathbf{r}) = -V_K(\mathbf{r}) + W_K(\mathbf{r}), \tag{2.37}$$

where the definition of $W_K(\mathbf{r})$ in eq. (2.2) is used. In other words, the constraint force in eq. (2.1) is given by $-\mu_K(\mathbf{r})$, i.e. the minus of the chemical potential when the segment concentration is restricted to $\{\phi_K(\mathbf{r})\}$.

If we are interested only in the full equilibrium state of the segment concentration distributions, the only constraint condition is the following local incompressible condition.

$$\sum_K \phi_K(\mathbf{r}) = 1. \tag{2.38}$$

In such a case, all the $\mu_K(\mathbf{r})$'s given by eq. (2.37) should be equal irrespective of the segment type K.

$$\mu_K(\mathbf{r}) = \mu(\mathbf{r}) \quad \text{for all } K. \tag{2.39}$$

Namely the constraint force for the local incompressible condition is given by $-\mu(\mathbf{r})$ which does not depend on the segment species. Precisely speaking, there can be a constant difference between $\mu_K(\mathbf{r})$'s because of the constant uncertainty in the self consistent field. Therefore, eq. (2.39) is a sufficient condition for μ_K to give the full equilibrium concentration distributions of the segments.

If some constraint conditions are given and the corresponding constraint forces are decided, the equilibrium state is achieved by obtaining the self consistent solutions of the set of equations eqs. (2.1), (2.14) (2.15) and (2.17) as is shown in fig. 2.1.

2.6 Ginzburg-Landau theory using Random Phase Approximation

The SCF theory is a accurate theory but it demands much computational time. Thus we propose a dynamic density functional theory which is not so accurate but can reduces the computational time. The method is based on the Ginzburg-Landau theory using the Random Phase Approximation (RPA). We will abbreviate this method as GRPA. The free energy model by Bohbot-Raviv and Wang is used, which is given by [11].

$$\begin{aligned}\mathcal{F}[\{\phi_i\}] &= \frac{1}{\beta}\int d\mathbf{r}\sum_i \frac{\phi_i(\mathbf{r})}{N_i}\ln\phi_i(\mathbf{r}) - \frac{1}{2\beta}\int d\mathbf{r}\sum_i \frac{1}{N_i\bar{\phi}_i}\delta\phi_i(\mathbf{r})\delta\phi_i(\mathbf{r}) \\ &\quad + \frac{1}{2\beta}\sum_{ij}\int d\mathbf{q} S_{ij}^{-1}(\mathbf{q})\delta\phi_i(\mathbf{q})\delta\phi_j(-\mathbf{q}),\end{aligned} \tag{2.40}$$

where the first term on the right-hand side is the Flory-Huggins entropy terms of the subchains. The second term is the minus of the second order term in the expansion of the entropy term (first term) in the segment density fluctuation $\delta\phi_i(\mathbf{r})$ given by

$$\delta\phi_i(\mathbf{r}) \equiv \phi_i(\mathbf{r}) - \bar{\phi}_i \tag{2.41}$$

where $\bar{\phi}_i$ is the average segment density in the system. The third term in eq.(2.40) is the Fourier transformation of the second order term in the expansion of the free energy in the segment density fluctuation $\delta\phi_i(\mathbf{r})$ of which the coefficient becomes the scattering function $S_{ij}^{-1}(\mathbf{q})$ between the segment density fluctuations of the i-th and j-th subchains. This second term is necessary to cancel the double counting of the second order terms in the expansion of the entropy term which is explicitly given by the third term. The scattering function $S_{ij}^{-1}(\mathbf{q})$ can be calculated by using the RPA for any polymer structures [12]. After calculating the third term in reciprocal space, it can be transformed to the term in reciprocal space by inverse Fourier transformation as

$$\begin{aligned}\mathcal{F}[\{\phi_i\}] &= \frac{1}{\beta}\sum_i^n \int d\mathbf{r}\frac{\phi_i(\mathbf{r})}{N_i}\ln\phi_i(\mathbf{r}) - \frac{1}{2\beta}\sum_i^n\int d\mathbf{r}\frac{1}{N_i\bar{\phi}_i}\delta\phi_i(\mathbf{r})\delta\phi_i(\mathbf{r}) \\ &\quad + \frac{1}{2}\int d\mathbf{r}\ \mathbf{x}^T(\mathbf{r})\mathbf{u}(\mathbf{r})\end{aligned} \tag{2.42}$$

where $\mathbf{x}(\mathbf{r})$ is a vector whose elements are the segment density fluctuations and $\mathbf{x}^T(\mathbf{r})$ is the transposed vector of $\mathbf{x}(\mathbf{r})$.

The chemical potential of the i-th subchain is given by the functional derivative of the free energy in terms of $\phi_i(\mathbf{r})$ under the incompressible condition as

$$\mu_i(\mathbf{r}) = \frac{\delta\mathcal{F}[\{\phi_i\}]}{\delta\phi_i(\mathbf{r})} \tag{2.43}$$

$$\begin{aligned}&= \frac{1}{\beta}\left\{\frac{\ln\phi_i(\mathbf{r})}{N_i} + \frac{1}{N_i} - \frac{\ln\phi_n(\mathbf{r})}{N_n} - \frac{1}{N_n} - \frac{1}{\bar{\phi}_i N_i}\delta\phi_i(\mathbf{r}) + \frac{1}{\bar{\phi}_n N_n}\delta\phi_n(\mathbf{r})\right\} \\ &\quad + u_i(\mathbf{r}).\end{aligned} \tag{2.44}$$

We can use $\mu_i(\mathbf{r})$ for dynamic density functional theory. GRPA uses the RPA thus it strictly can be used in the weak segregation region but it can reproduces a qualitative phase behavior of the polymer melts and blends. Thus GRPA is a useful method but available systems are restricted by boundary conditions due to the Fourier transformation.

2.7 Obstacles

It is important for polymeric materials to study the state of polyme melts in which obstacles exist.

SUSHI can introduce nano particles to systems in SCF calculations. Polymers can be existed in either inner side or outer side of a particle.

2.7.1 Soft particle dynamics

The SCF calculation cost of obstacles with hard interface is high. One idea to reduce the cost is to use soft interface obstacles. SUSHI was modified to introduce such a soft particle as followings[13].

Set the particle as a fixed shape solvent region which has the segment density gradient on the surface defined as

$$\phi_p(\mathbf{r}) = \frac{\tanh\{\alpha(r - |\mathbf{r} - \mathbf{r}_0|)\} + 1}{2}. \tag{2.45}$$

where α is the coefficient to set the depth of the surface, r is the radius of the paricle and $\mathbf{r}_0$ is the coordinate of the center of particle. Solve the static SCF calculation with fixed the shape of particle. Afetr the calculation, we can obtained the chemical potential at $\mathbf{r}_s$ on the surface as $\mu_p(\mathbf{r}_s)$ with the integration of the chemical potential within the soft region. Using this $\mu_p(\mathbf{r}_s)$, we can define the effective force to the paricle as

$$\mathbf{F}_p = D_e \int_S \nabla\mu_p(\mathbf{r}_s) \frac{\mathbf{r} - \mathbf{r}_0}{|\mathbf{r} - \mathbf{r}_0|} ds \tag{2.46}$$

where D_e is the effective diffusion constant of the particle. Using the force, we can move the particle and return to the static SCF calculation. By the iteration on those manner, we can simulate a particle dynamics.

2.8 Dynamical mean field method

The dynamical mean field method is a method to simulate diffusion dynamics of the segments driven by the gradient of the chemical potential, where the conformational entropy is taken into account through the path integral.

Let $\phi_K(\mathbf{r}, t)$ and $\mu_K(\mathbf{r}, t)$ be the segment concentration and the chemical potential at time t, then the time evolution equation of the segment density is in general assumed to be

$$\frac{\partial}{\partial t}\phi_K(\mathbf{r}, t) = \sum_{K'} \int d\mathbf{r}' \Lambda_{KK'}(\mathbf{r}, \mathbf{r}')\mu_{K'}(\mathbf{r}', t) + \xi_K(\mathbf{r}, t). \tag{2.47}$$

In order to integrate this time evolution equation, one has to evaluate the chemical potential $\mu_K(\mathbf{r})$ using eq. (2.37) where the contribution from the conformational entropy is calculated by solving the set of self consistent equations described in the preceding sections. Then, substituting the result into eq. (2.47), one can simulate the diffusion dynamics of the segment taking the chain conformational entropy into account. $\Lambda_{KK'}(\mathbf{r}, \mathbf{r}')$ accounts for the transmission rate of an external force acting on the K'-type segment at position $\mathbf{r}'$ to the K-type segment at position $\mathbf{r}$, and is called a non-local mobility. $\xi_K(\mathbf{r}, t)$ is the random noise due to the thermal fluctuation, which satisfies the following fluctuation-dissipation relation.

$$\langle \xi_K(\mathbf{r}, t)\xi_{K'}(\mathbf{r}', t')\rangle = 2k_B T \Lambda_{KK'}(\mathbf{r}, \mathbf{r}')\delta(t - t'). \tag{2.48}$$

In the dynamic mean field method adopted in SUSHI, we only consider the case where the change in the segment distribution is caused by the diffusion flux of segments that is proportional to the gradient of the chemical potential (Fick's law). In this case, eq. (2.47) reduces to the following simple form

$$\frac{\partial}{\partial t}\phi_K(\mathbf{r}, t) = \nabla \cdot [L_K(\mathbf{r}, t)\nabla\{\mu_K(\mathbf{r}, t) + \lambda(\mathbf{r}, t)\}] + \xi_K(\mathbf{r}, t), \tag{2.49}$$

where $L_K(\mathbf{r},t)$ expresses the mobility of the K-type segment, and $\lambda(\mathbf{r},t)$ is a Lagrange multiplier for the local incompressible condition. When the density of the K-type segments becomes very low, the dependence of $L_K(\mathbf{r},t)$ on the local segment density $\phi_K(\mathbf{r},t)$ should explicitly be taken into account as

$$L_K(\mathbf{r},t) = \frac{N}{k_BT} D_G \phi_K(\mathbf{r},t) \qquad (\phi_K \ll 1), \tag{2.50}$$

where D_G is the diffusion coefficient of the center-of-mass of a chain of the dilute component. Note that this diffusion constant changes according to the environment. There are two typical extreme limits for D_G.

1) For a K-type polymer dilutely dissolved into a solvent (Rouse Dynamics condition), D_G is expressed as

$$D_G = \frac{k_BT}{N_K\zeta} \tag{2.51}$$

and we obtain

$$L_K(\mathbf{r},t) = \frac{1}{\zeta}\phi_K(\mathbf{r},t) \qquad (\phi_K \ll 1). \tag{2.52}$$

2) For a K-type polymer in a polymer melt (Reptation Dynamics condition), D_G is given by

$$D_G = (\frac{a^2}{b_K^2})\frac{k_BT}{3N_K^2\zeta} \tag{2.53}$$

and then we obtain

$$L_K(\mathbf{r},t) = (\frac{a^2}{b_K^2})\frac{1}{3N_K\zeta}\phi_K(\mathbf{r},t) \qquad (\phi_K \ll 1), \tag{2.54}$$

where ζ is the viscosity of the matrix (the solvent or the melt) surrounding the tagged chain, and a is the characteristic length of the network formed by the surrounding entangled polymers (average distance between entangling points)[20],[21].

When the concentration of the K-type chain is high, the validity of these approximations are not guaranteed. However, these expressions in the dilute limit can still be used even for a high density polymer component as a phenomenological model.

The Lagrange multiplier $\lambda(\mathbf{r},t)$ is determined by the local incompressible condition, and is expressed as

$$\nabla\lambda(\mathbf{r},t) = -\frac{\sum_{K'} L_{K'}(\mathbf{r},t)\nabla\mu_{K'}(\mathbf{r},t)}{\sum_{K''} L_{K''}(\mathbf{r},t)}. \tag{2.55}$$

The thermal noise, $\xi_K(\mathbf{r},t)$ is assumed to be a Gaussian noise that satisfies

$$\langle \xi_K(\mathbf{r},t)\xi_{K'}(\mathbf{r}',t')\rangle = 2k_BT\mathcal{L}_{KK'}(\mathbf{r},t)\nabla^2\delta(\mathbf{r}-\mathbf{r}')\delta(t-t'), \tag{2.56}$$

where $\mathcal{L}_{KK'}(\mathbf{r},t)$ is given by the mobility coefficient $L_K(\mathbf{r},t)$ as

$$\mathcal{L}_{KK'}(\mathbf{r},t) = L_K(\mathbf{r},t)\delta_{KK'} + \frac{L_K(\mathbf{r},t)L_{K'}(\mathbf{r},t)}{\sum_{K''} L_{K''}(\mathbf{r},t)}. \tag{2.57}$$

In the dynamic mean field method, the self consistent equations are solved under the constraint that the concentration profiles of the segments are constrained to the given profiles $\{\phi_K(\mathbf{r})\}$ that change according to the time evolution. The actual numerical scheme for performing this procedure will be described in the next section.

The final structure obtained by the dynamic mean field calculation of eq. (2.49) should in principle be the same as that obtained by the static equilibrium calculation because both structures satisfy the condition eq. (2.39). However, the structure thus-obtained does not always give the minimum free energy but in many cases gives a local minimum of the free energy. For example, a dynamic simulation on the micro-phase separation of a block copolymer often results in a metastable state containing many defects, although the number of such defects can be controlled by changing the temperature. On the other hand, similar metastable states can usually be obtained in real experiments where the system is quenched from a high temperature uniform state to a coexisting state. In this sense, we can insist that the metastable state obtained by the

dynamic calculation correctly reflects the history of the system. In order to reproduce the dynamical process faithfully, inclusion of the noise term in eq. (2.49) may be important. However, such a noise term can usually be neglected for melts of long chains due to the strong suppression of the thermal fluctuation.

The dynamic calculation can also be performed using the Approximate Density Functional method. Since an analytic expression of the chemical potential is easily be obtained for this model (see eqs. (2.35) or (2.36)), computational cost is considerably lower than the dynamic mean field method combined with the self consistent field method. However, since conformational entropy of the chains is not correctly taken into consideration in the Approximate Density Functional method, the simulation result may be quantitatively less accurate.

2.8.1 Shear

An external flow can be introduced to the dynamics of K-type segment density. The dynamical equation (2.49) is extended by the itroduction of the velocity $v_K(\mathbf{r})$ of external flow as

$$\frac{\partial}{\partial t}\phi_K(\mathbf{r},t) = \nabla\cdot[L_K(\mathbf{r},t)\nabla\{\mu_K(\mathbf{r},t)+\lambda(\mathbf{r},t)\}] + \xi_K(\mathbf{r},t) - \nabla\{v_K(\mathbf{r})\phi_K(\mathbf{r})\}. \tag{2.58}$$

In regular mesh, The velocity of shear flow in parallel to XY plane and in the direction of X is given by

$$v_K(\mathbf{r}) = \big(\dot\gamma(r_y - r_{y_0}), 0, 0\big) \tag{2.59}$$

where $\dot\gamma$ is the shear rate

$$\dot\gamma = \frac{\partial v_x}{\partial y} \tag{2.60}$$

and r_{y_0} is the position where the shear rate is 0 in Y axis. Lees-Edwards boundary condition must be introduced in this calculation.

2.8.2 Compressible dynamics

The dynamics of segment density is available without the Lagrange multiplier $\lambda(\mathbf{r},t)$ in eq. (2.49) by the introduction of the compressibility κ [6]. In this method, the elastic energy term is introduced to the free energy equation (2.30) as

$$\begin{aligned}\mathcal{F}\big[\{\phi_K\},\{V_K\}\big] &= -k_{\mathrm{B}}T\sum_p M_p \ln \mathcal{Z}_p + \mathcal{W}\big[\{\phi_K\}\big] \\ &\quad -\sum_K \int d\mathbf{r} V_K(\mathbf{r})\phi_K(\mathbf{r}) \\ &\quad +k_{\mathrm{B}}T\sum_p M_p \ln M_p \\ &\quad +\frac{1}{2\kappa}\big(\sum_K \phi_K(\mathbf{r}) - 1\big)^2.\end{aligned} \tag{2.61}$$

The self consistent field equation (2.1) is also extended by the introduction as

$$V_K(\mathbf{r}) = W_K(\mathbf{r}) + V_c(\mathbf{r}) + \Big[\text{potential decided by incompressible condition}\Big] \tag{2.62}$$

where $V_c(\mathbf{r})$ is the elastic potential expressed by

$$V_c(\mathbf{r}) = \frac{1}{\kappa}\big(\sum_K \phi_K(\mathbf{r}) - 1\big). \tag{2.63}$$

The dynamics without $\lambda(\mathbf{r},t)$ but with the compressibility does not satisfy the complete incompressibility, i.e. $\sum_K \phi_k(\mathbf{r}) \neq 1$.

2.8.3 Handling of chemical reaction

One can perform a simulation on a chemical reaction by adding a chemical reaction term to the right-hand-side of the dynamic mean field equation eq. (2.49). For example, if we consider a First order reaction where the A species changes into B species with a reaction constant k, the explicit expression of the reaction term is given by

$$\frac{\partial \phi^{(A)}(\mathbf{r},t)}{\partial t} = \frac{\partial \phi^{(A)}(\mathbf{r},t)}{\partial t}\Big|_{diffusion} - k\phi^{(A)}(\mathbf{r},t) \tag{2.64}$$

$$\frac{\partial \phi^{(B)}(\mathbf{r},t)}{\partial t} = \frac{\partial \phi^{(B)}(\mathbf{r},t)}{\partial t}\Big|_{diffusion} + k\phi^{(A)}(\mathbf{r},t) \tag{2.65}$$

Here, the first term on the right-hand-side is the contribution from the diffusion of segments given by the right-hand-side of eq. (2.49). The second term is the contribution from the reaction. The actual simulation method will be described in the next section.

2.8.4 Hybrid method

The hybridization of the dynamic GRPA and the dynamic SCF method accelerates the dynamic SCF calculation without spoiling the accuracy of the SCF calculation [12].

In the hybridization, the chemical potential of the GRPA is corrected by the chemical potential of the SCF theory. We denote the chemical potentials of the hybrid and the dynamic GRPA method at a position $\mathbf{r}$ and at a time t by $\mu_i^{HYB}(\mathbf{r},t)$ and $\mu_i^{RAP}(\mathbf{r},t)$, respectively. The hybridized chemical potential is given by

$$\mu_i^{HYB}(\mathbf{r},t) = \mu_i^{RPA}(\mathbf{r},t) + \Delta\bar{\mu}_i(\mathbf{r}) \tag{2.66}$$

where $\Delta\bar{\mu}_i(\mathbf{r})$ is a correction term which is given by the difference between $\mu_i^{RAP}(\mathbf{r},t)$ and the chemical potential of the SCF theory $\mu_i^{SCF}(\mathbf{r},t_0)$ as

$$\Delta\bar{\mu}_i(\mathbf{r}) = \mu_i^{SCF}(\mathbf{r},t_0) - \mu_i^{RAP}(\mathbf{r},t_0). \tag{2.67}$$

The t_0 is the update time of the $\Delta\bar{\mu}_i(\mathbf{r})$ and is set longer than the time step of the dynamics scheme. Therefore the calculation speed is accelerated due to the GRPA calculation but the result is corrected by the SCF theory.

2.8.5 Hydrodynamic effects

Hydrodynamic effects can be introduced to the density functional theory [19]. Reader should read the detail in Muffin manual.

The Navier-Stokes equation to be solved is given by

$$\rho\frac{\partial \mathbf{v}(\mathbf{r},t)}{\partial t} = -\rho\{\mathbf{v}(\mathbf{r},t)\cdot\nabla\}\mathbf{v}(\mathbf{r},t) - \nabla p(\mathbf{r},t) + \nabla\{\eta(\mathbf{r},t)\nabla\cdot\mathbf{v}(\mathbf{r},t)\} - \sum_i \phi_i(\mathbf{r},t)\nabla\mu_i(\mathbf{r},t), \tag{2.68}$$

where $p(\mathbf{r},t)$ is a pressure、 $\eta(\mathbf{r},t)$ is a local viscosity, and the last term on the right-hand side is a volume force which couples the density functional theory and the hydrodynamics. The viscosity $\eta(\mathbf{r},t)$ is assumed to be the sum of each viscosity of the segment as

$$\eta(\mathbf{r},t) = \sum \eta_K\phi_K(\mathbf{r},t), \tag{2.69}$$

where η_K is assumed to be a viscosity of K-type segment and $\phi_K(\mathbf{r},t)$ is a volume fraction of K-type segment.

Introducing the incompressible conditions

$$\nabla\cdot\mathbf{v}(\mathbf{r},t) = 0, \tag{2.70}$$

and assuming the adiabatic condition

$$\frac{\partial \mathbf{v}(\mathbf{r},t)}{\partial t} = 0, \tag{2.71}$$

the Poisson equation for $p(\mathbf{r},t)$ is given by

$$\nabla^2 p(\mathbf{r},t) = \nabla\cdot[-\rho\{\mathbf{v}(\mathbf{r},t)\cdot\nabla\}\mathbf{v}(\mathbf{r},t) + \nabla\{\eta(\mathbf{r},t)\nabla\cdot\mathbf{v}(\mathbf{r},t)\} - \sum_i \phi_i(\mathbf{r},t)\nabla\mu_i(\mathbf{r},t).\} \tag{2.72}$$

Solving this equation and insert $p(\mathbf{r},t)$ to eq. (2.68), we can update $\mathbf{v}(\mathbf{r},t)$. The updated $\mathbf{v}(\mathbf{r},t)$ can be used for eq. (2.58) and the hydrodynamic effects are introduced to the dynamic density functional theory.

2.8.6 External electric field

Treatment of the weak effect of external electric field

When an external electric field is applied and the effect is weak, a simple modification can be introduced to the dynamics equation eq. (2.47) as follows [14, 15, 16, 17, 18] :

$$\frac{\partial}{\partial t}\phi_K(\mathbf{r},t) = L\nabla^2\mu_K(\mathbf{r}',t) + \alpha\frac{\partial^2}{\partial z^2}\phi_K(\mathbf{r},t), \tag{2.73}$$

where the constant mobility L is assumed to the all segments and the random noise term is neglected. The final term includes the effect of the external electric fields and is given by

$$\alpha = \frac{\epsilon_0\epsilon_1^2}{\bar{\epsilon}}E_0^2 vL, \tag{2.74}$$

where ϵ_0 is the dielectric constant of vacuum, E_0 is the amplitude of the applied electric field and v is the volume of a single polymer chain.

The detail is described as fllows [14].

We define a system under an external electric field whose direction is Z direction expressed as $\mathbf{E}_0 \equiv (0,0,E_0)$. Under the $\mathbf{E}_0$, the system causes a dielectric polarization then an electrostatic potential $\varphi(\mathbf{r})$ is generated by the dielectric polarization. Thus the effective electric field is given by

$$\mathbf{E}(\mathbf{r}) = \mathbf{E}_0 - \nabla\varphi(\mathbf{r}). \tag{2.75}$$

We also assume that the effect of the external electric field is weak and the local dielectric constant is given by

$$\begin{aligned}
\epsilon(\mathbf{r}) &\approx \bar{\epsilon} + \epsilon_1(\phi_A(\mathbf{r}) - f) && (2.76)\\
&\bar{\epsilon} = \epsilon|_{\phi_A=f} = \epsilon_A f + \epsilon_B(1-f) && (2.77)\\
&\epsilon_1 = (\partial\epsilon/\partial\phi_A)\,|_{\phi_A=f} = \epsilon_A - \epsilon_B. && (2.78)
\end{aligned}$$

The Poisson's equation $\nabla\cdot\epsilon(\mathbf{r})\mathbf{E}(\mathbf{r}) = 0$ must be satisfied. Thus

$$\begin{aligned}
&\nabla\cdot\{\bar{\epsilon} + \epsilon_1(\phi_A(\mathbf{r}) - f)\}\{\mathbf{E}_0 - \nabla\varphi(\mathbf{r})\} && (2.79)\\
&\approx \nabla\cdot\{-\bar{\epsilon}\nabla\varphi(\mathbf{r}) + \epsilon_1\phi_A(\mathbf{r})\mathbf{E}_0\} = 0, && (2.80)
\end{aligned}$$

where the heigher order term is neglected. Therefore

$$\bar{\epsilon}\nabla^2\varphi(\mathbf{r}) = \epsilon_1 E_0\nabla_Z\phi_A(\mathbf{r}). \tag{2.81}$$

The chemical potential caused by the electrostatic energy to $\phi_A(\mathbf{r})$ is derived as

$$\mu_{el}(\mathbf{r}) = -v\frac{\delta}{\delta\phi_A(\mathbf{r})}\Big[\frac{\epsilon_0}{2}\int d\mathbf{r}\frac{\mathbf{E}(\mathbf{r})^2\epsilon(\mathbf{r})}{4\pi}\Big], \tag{2.82}$$

where the Legendre transformation is used to change the independent variable from $\mathbf{E}(\mathbf{r})$ to $\phi(\mathbf{r})$, and the sign in front of the right hand side is set to minus.

Using the eqs. (2.75), (2.76) and (2.81), $\nabla^2\mu_{el}(\mathbf{r})$ is given by

$$\begin{aligned}
\nabla^2\mu_{el}(\mathbf{r}) = &\quad -\frac{v\epsilon_0}{8\pi}\nabla^2\epsilon_1(\mathbf{E}_0^2 - 2E_0\nabla_Z\varphi(\mathbf{r}) + (\nabla\varphi(\mathbf{r}))^2) && (2.83)\\
\approx &\quad \frac{v\epsilon_0\epsilon_1 E_0}{4\pi}\nabla_Z\{\nabla^2\varphi(\mathbf{r})\} && (2.84)\\
= &\quad \frac{v\epsilon_0\epsilon_1^2 E_0^2}{4\pi\bar{\epsilon}}\nabla_Z^2\phi_A(\mathbf{r}). && (2.85)
\end{aligned}$$

Multiplication of the mobility L to the right-hand side of the final equation gives eq. (2.74).

General treatment of the effect of external electric field

Here we introduce the external electric field faithfully. Under an external electric field $E_0(\mathbf{r})$, the following modified Poisson equation should be satisfied.

$$\nabla \cdot \epsilon(\mathbf{r})\{\nabla U'(\mathbf{r}) - E_0(\mathbf{r})\} = -\rho(\mathbf{r}), \tag{2.86}$$

where $\epsilon(\mathbf{r})$ is the local dielectric constant and $\rho(\mathbf{r})$ is the local charge. The $U'(\mathbf{r})$ is the induced electrostatic potential by the external electric field and the local charge.

We assume that the local dielectric constant $\epsilon(\mathbf{r})$ and the local charge $\rho(\mathbf{r})$ are given by

$$\epsilon(\mathbf{r}) = \epsilon_0 \sum_K \epsilon_K \phi_K(\mathbf{r}), \tag{2.87}$$

$$\rho(\mathbf{r}) = \sum_K \rho_K \phi_K(\mathbf{r}), \tag{2.88}$$

where ϵ_K and ρ_K are the relative dielectric constant and charge of K-type segment per volume, respectively.

Equation (2.86) can be modified as

$$\nabla\epsilon(\mathbf{r})\nabla U'(\mathbf{r}) = -\rho(\mathbf{r}) + \nabla \cdot \epsilon(\mathbf{r})E_0(\mathbf{r}). \tag{2.89}$$

By solving this equation, we can get the induced electrostatic potential $U'(\mathbf{r})$. The effective electrostatic potential $U(\mathbf{r})$ and the effective electric field $E(\mathbf{r})$ to the system can be given by

$$U(\mathbf{r}) = U(\mathbf{r})' - \int E_0(\mathbf{r})d\mathbf{r} \tag{2.90}$$

$$E(\mathbf{r}) = -\nabla U(\mathbf{r})' + E_0(\mathbf{r}) \tag{2.91}$$

These effective term can be involved to the SCF calculation through the following free energy equation.

$$F' = F + \frac{1}{2}\int U(\mathbf{r})\rho(\mathbf{r})dv - \frac{1}{2}\int \epsilon(\mathbf{r})E^2(\mathbf{r})dv, \tag{2.92}$$

where, in the right-hand side, F is the free energy without the electrostatic effect, the second term is the effect of the local charge, and the third term is the effect of the electric field. The self-consistent field $V(\mathbf{r})$ is modified by using this free energy equaiton as follow.

$$V'_K(\mathbf{r}) = W'_K(\mathbf{r}) - \mu_K(\mathbf{r}), \tag{2.93}$$

where $W'_K(\mathbf{r})$ is the modified interaction energies concerning to the segments given by

$$W'_K(\mathbf{r}) = W_K(\mathbf{r}) + \rho_K U(\mathbf{r}) - \epsilon_K E^2_K(\mathbf{r}). \tag{2.94}$$

2.9 Method of calculation

2.9.1 Branch structure

Branch structure is one of the important features in the coarse-grained polymer model. In case of the SCF simulation, it is crucial how this branch structure is modeled. In the calculation by using SUSHI, as explained in Sections 2.1 and 2.2 before, a branched polymer is modeled as a set of subchains each of which are composed of a specific monomer sequence and are connected at junctions (junction points). Here, the subchain is defined as a linear chain not only composed of a single type of segment but also composed of several types of segments with arbitrary monomer sequences such as random, alternative, tapered etc. The junction point is defined as the point that connects these subchains. The junction point is assumed to have a vanishing specific volume. See fig. 2.4.

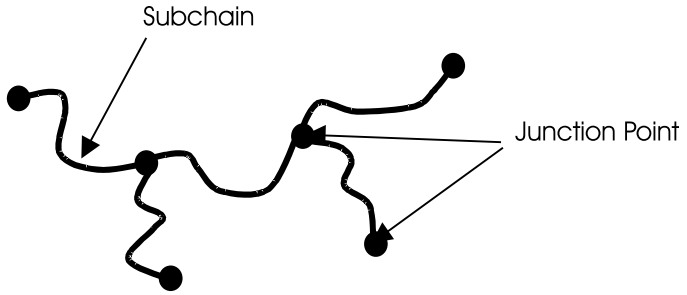


Figure 2.4: Model of branched polymer

In SUSHI, a simple input method is offered for three types of typical branch structures that are shown in fig. 2.5.

1) Linear-type block copolymer

2) Comb-type block copolymer

3) Star-type block copolymer

For general branch structures other than these three types, the network structure composed of the subchains should be specified directly using a more general method. The details of such a general method will be explained in Chapter 5.

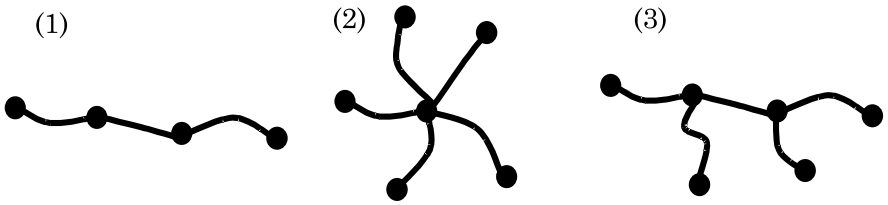


Figure 2.5: Simple branched block copolymer: (1) Linear-type, (2) Star-type, and (3) Comb-type

2.9.2 Calculation of path integrals

As is shown in Section 2.2, the statistical probability distribution of the conformation of the whole chain is obtained by numerically solving the evolution equation for the path integral $Q_K(s,\mathbf{r})$, eq. (2.21), for each of the subchains. Here, the segments are labeled by a continuous variable s instead of the discrete index i used in Section 2.1. The contributions from the other subchains connected to the junction point $s=0$ are reflected through the following initial condition.

$$Q_K(0,\mathbf{r}) = q_0(\mathbf{r}). \tag{2.95}$$

By taking the internal states (multi-states) introduced in Section 2.3 into account, the evolution equation eq. (2.29) is extended to the following equation.

$$Q_K(s+ds,\mathbf{r}) = \exp\bigl[-\beta r_K V_K(\mathbf{r})ds/2\bigr]\sum_{K'} T_{KK'}(s)\Bigl(1+\frac{b_{K'}^2}{6}\mathcal{L}ds\Bigr)\Bigl(\exp\bigl[-\beta r_{K'} V_{K'}(\mathbf{r})ds/2\bigr]Q_{K'}(s,\mathbf{r})\Bigr), \tag{2.96}$$

where $Q_K(s,\mathbf{r})$ is the statistical weight of a subchain of length s starting ($s=0$) from an arbitrary position with an arbitrary internal state and ending at position $\mathbf{r}$ with the internal state K, and $V_K(\mathbf{r})$ is the self-consistent field acting on the segment with the internal state K at position $\mathbf{r}$. $T_{KK'}(s)$ is the state transition probability defined as the probability of finding the $s+ds$ segment in the internal state K provided that the s segment was found in the internal state K'. $\mathcal{L}$ is the Laplace operator ∇^2, and b_K is the effective bond length that is equivalent to the size of the segment. We also introduced the "specific volume" r_K, which

is defined as the dimensionless segment molar volume normalized by a standard segment volume. For the sake of the convenience in the numerical calculations, the evolution equation eq. (2.96) has been transformed into the integral form that corresponds to the recurrence formula eq. (2.5). In order to perform numerical calculation, we descretize the continuous segment index s and the spatial coordinate $\mathbf{r}$. The discretization of s is realized by substituting a finite value into ds in the evolution equation eq. (2.96). The discretization of $\mathbf{r}$ is performed by introducing a spatial mesh and a corresponding discrete Laplace operator $\mathcal{L}$ on the mesh. These methods will be described in the next section.

2.9.3 Various spatial mesh structures

Since the main part of the calculation performed in SUSHI is a numerical integration of the partial differential equations for the path integrals and the segment density fields, a discretization of the space is required to calculate these fields of physical quantities. In SUSHI, various types of spatial meshes (spatial lattices) are prepared according to the types of the target phenomena.

The targets of the SCF calculation can be categorized into the following two classes.

1) Highly symmetric structures such as a single interface or a micelle with the spherical or the rod-like structure.

2) Disordered domain structures formed in the phase separation processes.

For the former symmetric case, it is computationally efficient to choose an appropriate coordinate system for the symmetry of the problem such as the polar coordinate or the cylindrical coordinate. On the other hand, for the latter case, usually cubic lattice or a cartesian coordinate system is used. The spatial meshes that are now available in SUSHI are listed below.

1) 1,2 and 3 dimensional regular meshes (RegularMesh, fig. 2.6)
These are cartesian coordinate systems with their axes along the x, y and z directions. These meshes are used for the calculations of the general structures with the lowest symmetry, such as an irregular structure.

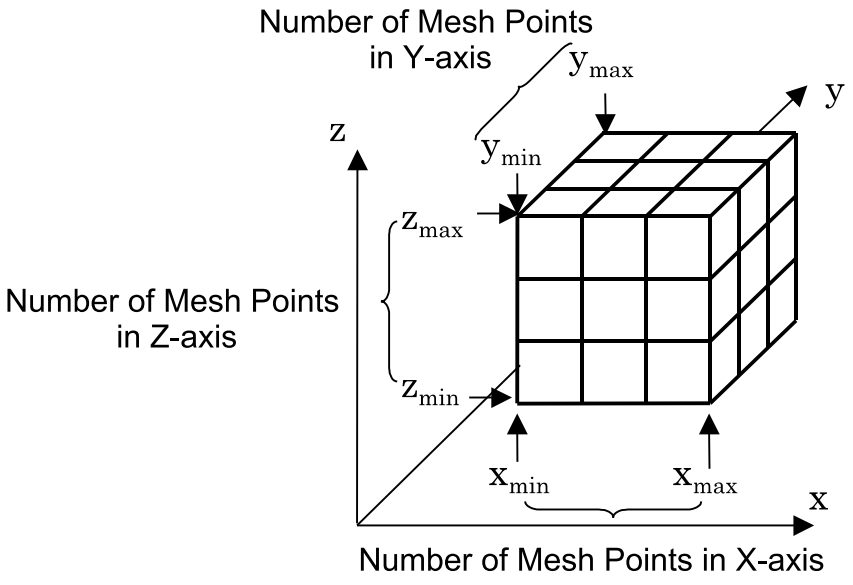


Figure 2.6: 3D Regular mesh: The mesh sizes along the 3 axes are all fixed.

2) 1,2 and 3 dimensional rectangular meshes (RectangularMesh, fig. 2.7)
These are cartesian coordinate systems with the coordinate axes parallel to the x, y, and z directions but with non-uniform grid sizes. These are suitable for the calculation of the systems in which physical quantity changes abruptly only in a small region, such as a domain structures divided by a flat interface. The mesh sizes shown in fig. 2.7 can be non-uniform.

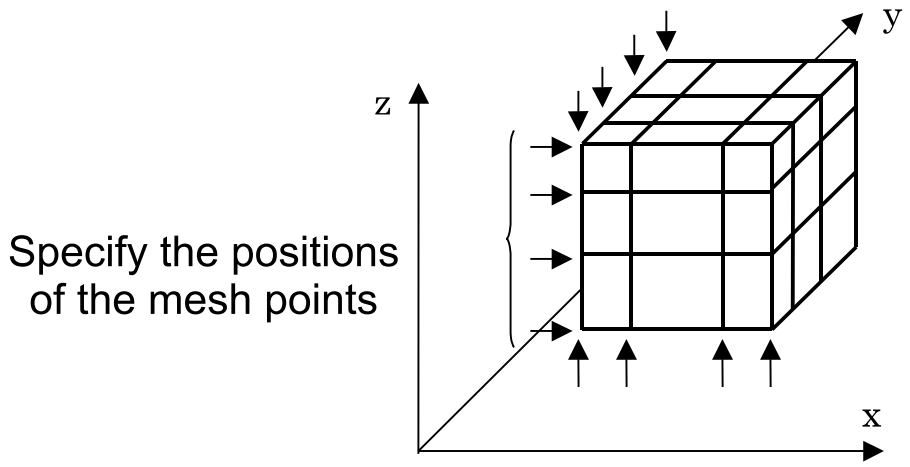


Figure 2.7: Three dimensional rectangular mesh. The mesh width can be non-uniform.

3) Polar-coordinate mesh (SphericalMesh, fig. 2.8)
3-dimensional polar coordinate system with the spherical symmetry is described using only the radial coordinate r. It is used for the calculation of spherically symmetric systems such as a spherical micelle and a spherical vesicle. The mesh width should be uniform.

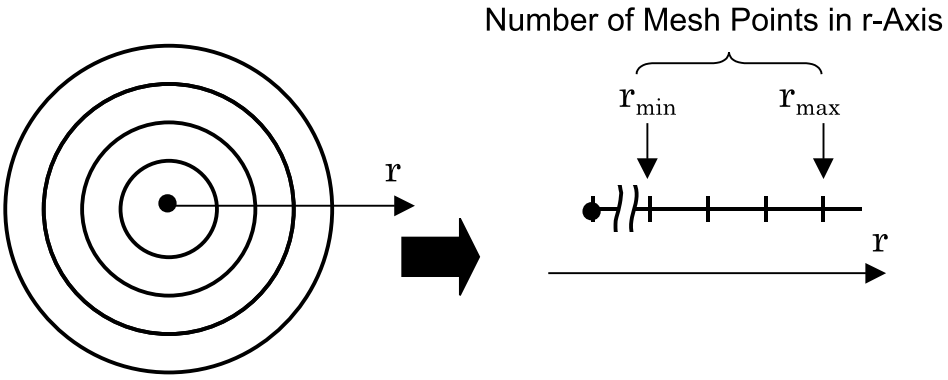


Figure 2.8: 3-dimensional polar coordinate system

4) Cylindrical mesh (CylindricalMesh,fig. 2.9)
3-dimensional cylindrical coordinate with the uniaxial symmetry is described using the radial coordinate r and the coordinate along its axis h. It is suitable for the calculations of bodies of revolution, such as a cylindrical micelle. The mesh width should be uniform.

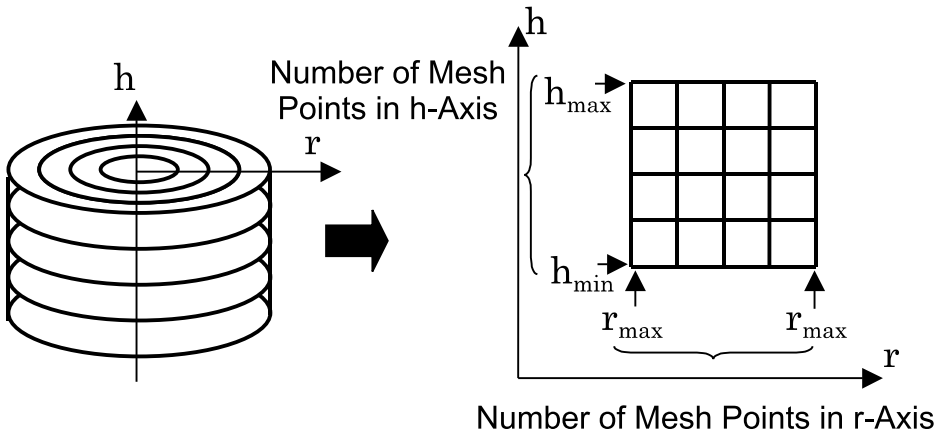


Figure 2.9: 3-dimensional cylindrical mesh: The mesh width is fixed and uniform.

For all of these meshes except for the rectangular mesh, the mesh width is indirectly specified by giving the side length of the simulation box in each coordinate axis and the total number of cells separated by the mesh points along the direction. In the rectangular mesh, the width is specified by directly specifying the positions of all the mesh points.

2.9.4 Boundary condition

Conceptually, the boundary conditions used in SUSHI can be classified into the following two categories.

1) Geometrical boundary conditions

2) Physical boundary conditions

The meaning of these categories are explained below.

1) Geometrical boundary conditions
   This boundary condition expresses the geometric (or topological) property of the system. For example, a 2-dimensional rectangular mesh system can be regarded as a rectangular-shaped system (non-period system) in contact with the external reservoir at the boundaries, or it can be regarded as a surface of a torus (periodic system) with the periodic boundary conditions where the two boundaries on both ends of the system are identified. As is shown in fig. 2.10, in SUSHI, these boundary conditions can be specified for each coordinate axis independently.

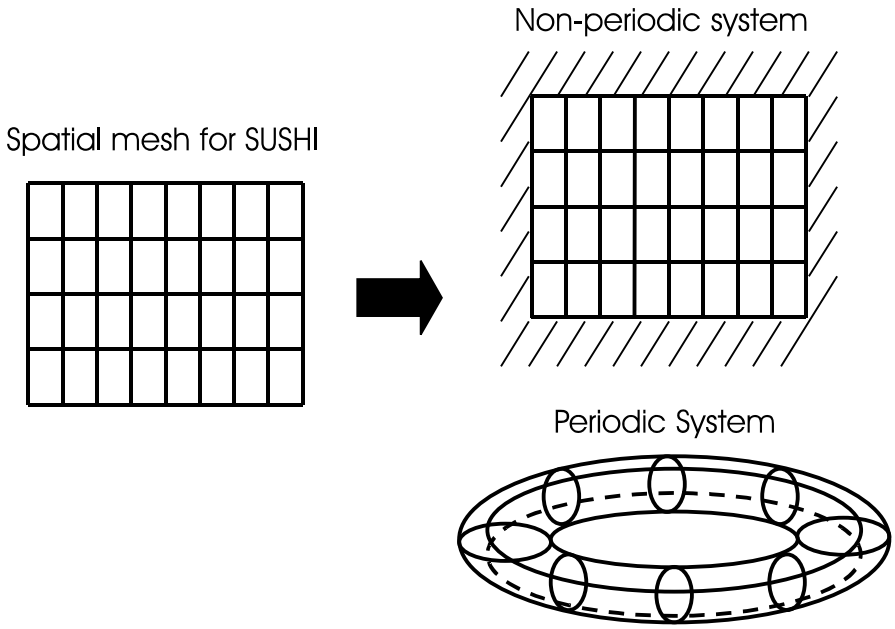


Figure 2.10: Explanation of the geometric boundary condition

Different treatments are used for the lattice points on a boundary depending on whether the boundary is a periodic boundary or a non-periodic boundary. For an axis with the periodic boundary condition, both ends of this axis is identical, while for an axis with non-periodic boundary condition, these two points are regarded as distinct.

2) Physical boundary conditions
Physical boundary conditions are the boundary conditions to specify the behavior of the field of the physical quantity on the boundary of a system. There are two types of the physical boundary condition.

   1) Neumann boundary condition :The value of the first derivative of the physical quantity is fixed.
   2) Dirichlet boundary condition :The value of the physical quantity itself is fixed.

   For example, a diffusion field faced to a non-penetrable wall can be regarded as a Neumann boundary at which the first derivative of the field is vanishing due to the conditions of vanishing flux perpendicular to the wall. When we consider a temperature distribution field is faced to a heat-conductive wall with a constant temperature, the wall should be treated as a Dirichlet boundary where the temperature is fixed. Possible combinations of the geometric boundary condition and the physical boundary condition are listed in table 2.1.

Table 2.1: Possible combinations of the boundary conditions

Geometric Boundary	Physical Boundary
Non-Periodic	Neumann Dirichlet
Periodic	Periodic

In the path integral calculation, the Dirichlet condition with the vanishing boundary value is used for the impenetrable solid wall. On the other hand, in the time evolution equation of the segment density field, such a solid wall is treated as a Neumann boundary with vanishing gradient of the segment density field. Therefore, a single geometrical boundary may be regarded as different physical boundaries for the path integral and for the segment density field. All the combinations listed in table 2.1 are possible for the regular mesh and the rectangular mesh, while only the non-periodic geometrical boundary condition is allowed for

the radial coordinate for the polar-coordinate mesh and the cylinder coordinate mesh, where the origin of the radial coordinate should always be a Neumann boundary due to the symmetry.

2.9.5 Discrete Laplace operator

The explicit form of the discrete Laplace operator $\mathcal{L}$ introduced in Section 2.9.2 depends on the mesh type. Even for a single type of mesh, several forms of the discrete Laplace operator can be defined. The scalar field in 3-dimensional space is expressed by a set of values $\{A(i,j,k)\}$ given at the mesh points specified by the combination of three integers (i,j,k). When the discrete Laplace operator is operated on a scalar field $A(\mathbf{r})$, the value of the resulting field $\mathcal{L}A(\mathbf{r})$ at the mesh point (i,j,k) is in general given by

$$\mathcal{L}A(i,j,k) = \sum_{l,m,n=0 \text{ or } \pm 1} C(l,m,n)A(i+l,j+m,k+n) \tag{2.97}$$

using the values of $A(\mathbf{r})$ at up to 27 nearest mesh points to the mesh point (i,j,k). Here, $C(l,m,n)$ is a coefficient depending on the mesh type and the method of discretization of the Laplace operator. Except for the regular mesh, the coefficient $C(l,m,n)$ depends on the position (i,j,k) due to the non-uniform nature of the mesh. In the following, examples of the set of values of the coefficient $C(l,m,n)$ are given for various types of the space meshes.

1) Regular Mesh
As mentioned in Section 2.9.3, all the mesh widths along each axis are the same (Δx). In this case, the following four types of discretized Laplacian can be used. (The mesh widths in all the directions are temporarily be assumed to be the same value Δx for simplicity.)

   i) 1NN-P:The second derivative is approximated by a central difference using consecutive three mesh points in each of the directions x, y and z. For the three dimensional system, the central mesh point and the six nearest neighbor mesh points are used. For one and two dimensional systems, the discretized Laplace operators are identical to the projections of the three-dimensional discrete Laplacian onto one and two dimensions.

   ii) 2NN-NP:To improve the isotropy, second nearest neighbor points are included in the difference scheme for the two and three-dimensional discrete Laplace operators. Although the one-dimensional 2NN-NP Laplace operator coincides with the projection of the higher-dimensional 2NN-NP operator, the two-dimensional one does not coincide with the projection of the three-dimensional 2NN-NP operator.

   iii) 2NN-P:As in the case of 2NN-NP, the nearest neighbors and the next nearest neighbors are also used but the coefficients are adjusted so that the two-dimensional discretized Laplace operator coincides with the projection of the three-dimensional operator. Both one and three-dimensional operators are the same as those in 2NN-NP case.

   iv) 3NN-P:The isotropy is more improved for the three dimensional operator by using 27 mesh points up to the third nearest neighbor mesh points. Both one and two dimensional operators coincide with the projections of the three-dimensional one.

   The actual values of the coefficients are listed in table 2.2. "Center" means the coefficient of the central mesh point, "NN", "NNN" and "NNNN" mean the coefficients of the "nearest neighbor", "next (second) nearest neighbor" and "next-next (third) nearest neighbor" mesh points, respectively.

2) Rectangular mesh
In the rectangular mesh, the indices i, j, and k specify the coordinates of the mesh point (denoting x_i, y_j, and z_k, respectively) in the x, y and z directions. In the current version of SUSHI, a method of approximating the second derivative by a central difference using three consecutive mesh points in each of the x, y and z directions is available. In this case, the value of $\mathcal{L}A(\mathbf{r})$ at the mesh point (i,j,k) is expressed as follows.

$$\mathcal{L}A(i,j,k) = \sum_{l^2+m^2+n^2=0 \text{ or } 1} C(i,j,k;l,m,n)A(i+l,j+m,k+n) \tag{2.98}$$

   If the surrounding mesh width of given mesh point (i,j,k) is expressed as

$$\Delta x^{(+)} \equiv x_{i+1} - x_i, \quad \Delta x^{(-)} \equiv x_i - x_{i-1}, \quad \Delta x \equiv (x_{i+1} - x_{i-1})/2,$$

Table 2.2: The actual numerical values of the coefficients in the various types of the discretized Laplace operator defined on a regular mesh

Type		Center ($\times\Delta x^{-2}$)	NN ($\times\Delta x^{-2}$)	NNN ($\times\Delta x^{-2}$)	NNNN ($\times\Delta x^{-2}$)
1NN-P	1D	-2	1	0	0
	2D	-4	1	0	0
	3D	-6	1	0	0
2NN-NP	1D	-2	1	0	0
	2D	-3	1/2	1/4	0
	3D	$-9/2$	1/2	1/8	0
2NN-P	1D	-2	1	0	0
	2D	$-7/2$	3/4	1/8	0
	3D	$-9/2$	1/2	1/8	0
3NN-P	1D	-2	1	0	0
	2D	$-34/11$	6/11	5/22	0
	3D	$-40/11$	3/11	3/22	1/22

$$\Delta y^{(+)} \equiv y_{j+1} - y_j, \quad \Delta y^{(-)} \equiv y_j - y_{j-1}, \quad \Delta y \equiv (y_{j+1} - y_{j-1})/2,$$
$$\Delta z^{(+)} \equiv z_{k+1} - z_k, \quad \Delta z^{(-)} \equiv z_k - z_{k-1}, \quad \Delta z \equiv (z_{k+1} - z_{k-1})/2,$$

the values of the coefficients $C(i,j,k;l,m,n)$ are listed in table 2.3. The one and the two dimensional operators coincide the projections of the three dimensional one. This discretized Laplacian operator reduces to the 1 NN-P operator for a regular mesh, when the mesh width is chosen to be uniform.

Table 2.3: The values of the coefficients of the discretized Laplacian operator defined on a rectangular mesh

(l,m,n)	$C(i,j,k;l,m,n)$
$(1,0,0)$	$1/(\Delta x^{(+)}\Delta x)$
$(-1,0,0)$	$1/(\Delta x^{(-)}\Delta x)$
$(0,1,0)$	$1/(\Delta y^{(+)}\Delta y)$
$(0,-1,0)$	$1/(\Delta y^{(-)}\Delta y)$
$(0,0,1)$	$1/(\Delta z^{(+)}\Delta z)$
$(0,0,-1)$	$1/(\Delta z^{(-)}\Delta z)$
$(0,0,0)$	$-2/(\Delta x^{(+)}\Delta x^{(-)}) - 2/(\Delta y^{(+)}\Delta y^{(-)}) - 2/(\Delta z^{(+)}\Delta z^{(-)})$

3) Polar coordinate mesh and Cylindrical mesh

i) Polar coordinate mesh and one-dimensional cylindrical coordinate mesh.
In these cases, as the coordinate system is essentially one-dimensional with the radial coordinate r, the 1NN-P type is adopted for the descretization of the coordinate r.

ii) Two-dimensional cylindrical mesh
The mesh is a two-dimensional mesh with the radial coordinate r and the coordinate h in the direction parallel to the axis of revolution. For the diecretized Laplacian operator, the 2NN-NP type discretization is used.

2.9.6 Treatment of grafted chains

In SUSHI, any junction points of a polymer chain can be grafted to arbitrary mesh point or to an arbitrary region of the mesh (set of mesh points). For a subchain whose end segment $s = 0$ is grafted to the point $\mathbf{r}$, its path integration is calculated using the initial condition $q_0(\mathbf{r}_0) = \delta(\mathbf{r}_0 - \mathbf{r})$ in eq. (2.23). When a polymer

is grafted to a certain region, $q_0(\mathbf{r}_0)$ takes the value unity inside of the region and vanishing outside the region so that the specified junction can exist only in the specified region. The junction points thus grafted can generally redistribute non-uniformly inside of the domain. That is, an equilibrium distribution of the graft points is restricted within the specified region. In the input data for SUSHI, one can graft a junction point by specifying the index of the junction point and the coordinates of mesh points to which the junction is grafted. Details will be described in Chapter 5.

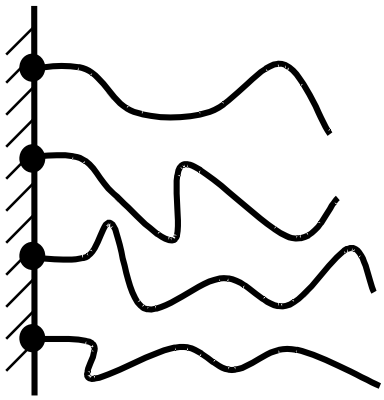


Figure 2.11: Model of grafted chains

2.9.7 External field

In SUSHI, the effect of short range interactions between a solid wall and a polymer segment can be specified by using interaction parameter χ_{KS} between a K-type segment and a wall (S). This extra interaction is added to the self consistent field $V_K(\mathbf{r})$ only on the nearest neighbor mesh point to the wall surface as

$$V_K(\mathbf{r}) = W'_K(\mathbf{r}) - \mu(\mathbf{r}) \tag{2.99}$$

$$W'_K(\mathbf{r}) = W_K(\mathbf{r}) + \chi_{KS}\phi(\mathbf{r}')/\mathcal{Y}(\mathbf{r}'), \tag{2.100}$$

where $\mathbf{r}'$ means the position at the nearest neighbor mesh point to the wall surface and $\mathcal{Y}(\mathbf{r})$ is the yacobian of the cell at the mesh point.

2.9.8 Techniques for realizing statistical ensembles and calculating segment densities

SUSHI offers the following two kinds of statistical ensembles for which the path integral calculation is performed.

1) Canonical ensemble where the temperature and the total numbers of molecules, chains and solvents in the system are kept constant.

2) Grand canonical ensemble where the temperature is kept constant and the system is assumed to contact with a reservoir of the particles with constant chemical potential.

The difference between these two kinds of statistical ensembles is reflected in the normalization conditions for the calculation of the segment density distributions using the path integrals. How to calculate the normalization constants is described in eqs. (2.10) and (2.11) or in eqs. (2.19) and (2.20). Here, we re-describe the formula for the segment density distribution (eq. (2.18)) and the formula for the normalization factor. Here we have introduced the specific volume r_K of the K-type segment. (The segment specific volume appears only in the normalization constant for the case of the grand canonical ensemble) The formula for the segment density distribution is given by

$$\phi_r^{(p)}(\mathbf{r}) = C^{(p)} \sum_i \int d\mathbf{r}_0 \int d\mathbf{r}_N q_0(\mathbf{r}_0) Q_K(0, \mathbf{r}_0; i, \mathbf{r}) Q_K(i, \mathbf{r}; N, \mathbf{r}_N) q_N(\mathbf{r}_N). \tag{2.101}$$

The normalization factor in the canonical ensemble case is given by

$$C^{(p)} = \frac{M^{(p)}}{\int d\mathbf{r}_0 \int d\mathbf{r}_N q_0(\mathbf{r}_0) Q_K(0, \mathbf{r}_0; N, \mathbf{r}_N) q_N(\mathbf{r}_N)} , \tag{2.102}$$

while for the grand canonical ensemble $C^{(p)}$ is as follows

$$C^{(p)} = \frac{\phi_p^{(\mathrm{bulk})}}{\sum_{r''} r_{K_{r''}^{(p)}} N_{r''}^{(p)}} \exp\Big[\sum_{r'} r_{K_{r'}^{(p)}} N_{r'}^{(p)} W_{K_{r'}^{(p)}}^{(\mathrm{bulk})}\Big]. \tag{2.103}$$

In SUSHI, one can specify the statistical ensemble (the canonical ensemble or the grand canonical ensemble) that should be applied for each chain type. For example, when a solid wall onto which a polymer brush is grafted is dipped in a solvent that contacts with a reservoir of particles, one has to use appropriate ensembles depending on the condition imposed on the chains and the solvent, respectively. The statistical ensemble for the grafted polymers should be the canonical ensemble while that for the solvent should be the grand canonical ensemble.

2.9.9 Static calculation

Static calculation is a procedure for finding a set of self consistent solutions for $\{V_K(\mathbf{r})\}$ and $\{\phi_K(\mathbf{r})\}$ that satisfy eqs. (2.1), (2.14), (2.15), (2.17), the local incompressible condition eq. (2.38), and the constraint force given by eq. (2.39) simultaneously. These solutions are numerically obtained in SUSHI using the following iteration scheme.

First, the path integral is calculated using eq. (2.96) and a suitable initial distribution for $V_K(\mathbf{r})$. Then, the segment density distribution is calculated using eq. (2.101). The normalization constant C_K for the canonical ensemble and that for the grand canonical ensemble are given by eqs. (2.102) and (2.103), respectively. Next, using the $\phi_K(\mathbf{r})$ thus obtained, the two fields $W_K(\mathbf{r})$ and $\mu_K(\mathbf{r})$ in eq. (2.37) are updated using the following iterative scheme. (the initial value for $W_K(\mathbf{r})$ is set to 0)

$$W_K(\mathbf{r}) \longrightarrow W_K(\mathbf{r}) + \mathrm{constW} \times \Big(\sum_{K'} \chi_{KK'} \phi_{K'}(\mathbf{r}) - W_K(\mathbf{r})\Big) \tag{2.104}$$

$$\mu_K(\mathbf{r}) \longrightarrow \begin{cases} \mu_A(\mathbf{r}) - \mathrm{constV} \times \Big(1 - \sum_{K'} \phi_{K'}(\mathbf{r})\Big) & \text{for } K = A \\ \mu_K(\mathbf{r}) - \mathrm{constV} \times \Big(\mu_K(\mathbf{r}) - \mu_A(\mathbf{r})\Big) & \text{for } K \neq A \end{cases} \tag{2.105}$$

For a multi-component system with segment species $K = A, B, C, \cdots$, the first formula in eq. (2.105) is applied to the first segment species A, and the second formula is for the other segment species $B, C, \cdots$. The two parameters which appear in these formulae, constW and constV, are appropriately chosen constants between 0.0 and 1.0, and should be specified in the input parameter list for SUSHI. (When the iteration scheme is diverging or is slowly converging, decreasing these parameters may improve the convergence.) By using the self consistent field $V_K(\mathbf{r})$ obtained from the updated $\mu_K(\mathbf{r})$ and $W_K(\mathbf{r})$ through the relation $V_K(\mathbf{r}) = W_K(\mathbf{r}) - \mu_K(\mathbf{r})$, the path integral is recalculated. This iteration procedure is repeated until the maximum of the updated amounts of $V_K(\mathbf{r})$ and $W_K(\mathbf{r})$ during a single iteration step becomes less than a certain error level and the incompressible condition $\sum_K \phi_K(\mathbf{r}) = 1$ is fulfilled within the same error level. This error level is also specified in the input parameter list for SUSHI.

2.9.10 Dynamic calculation

As was described in Section 2.8, solving the time evolution equation for the segment density, eq. (2.49)

$$\frac{\partial}{\partial t}\phi_K(\mathbf{r}, t) = \nabla \cdot [L_K(\mathbf{r}, t) \nabla \{\mu_K(\mathbf{r}, t) + \lambda(\mathbf{r}, t)\}] + \xi_K(\mathbf{r}, t) \tag{2.106}$$

and the self consistent equation with the restrictions on $\{\phi_K(\mathbf{r})\}$ simultaneously, the diffusion dynamics of the segment density fields is simulated taking the conformational entropy of the chains into account. The initial value of the segment distributions $\{\phi_K(\mathbf{r})\}$ at each mesh point is assumed to be a Gaussian random variable (Its standard deviation should be given as an input parameter), and the mobility coefficient $L_K(\mathbf{r}, t)$ can be assumed as follows.

1) Assuming a constant value of unity for all the segment species. This choice is suitable for melts that is composed of polymers with almost the same polymerization indices and is free from any dilute components.

2) Assuming $L_K(\mathbf{r},t)$ to be dependent on $\phi(\mathbf{r},t)$.

   Giving the constant parameter L_0 for each segment species, the mobility coefficients are assumed to have the form:

   Rouse Dynamics condition

$$L_K(\mathbf{r},t) = L_0\phi_K(\mathbf{r},t) \tag{2.107}$$

   Reptation Dynamics condition

$$L_K(\mathbf{r},t) = \frac{L_0}{N^{(total)}}\phi_K(\mathbf{r},t) \tag{2.108}$$

   Here, $N^{(total)}$ is the total number of segments in the chain to which the target segments belong. The validity of the formula for the Reptation Dynamics condition is guaranteed for at least linear homopolymer melts. In this case with segment density dependent mobility coefficients, the noise term $\xi_K(\mathbf{r},t)$ in eq. (2.106) is neglected.

At each step of the time evolution, the self consistent equation is solved under the condition that the segment density distribution calculated from the path integral should equal to $\{\phi_K(\mathbf{r})\}$ obtained by the integration of the evolution equation. Then the next time step evolution is performed using $\{\mu_K(\mathbf{r})\}$ obtained by the following procedure.

In order to complete the above scheme, one has to evaluate the chemical potential $\mu_K(\mathbf{r})$ that corresponds to a given segment density distribution $\phi_K(\mathbf{r})$ (hereafter it is denoted as $\phi_K^{\rm target}(\mathbf{r})$.). This procedure is "an equilibration under the constraint on the segment density", and is solved by an iterative method. First, we initialize $W_K(\mathbf{r})$ and $\mu_K(\mathbf{r})$ with 0. Then, we use the following iteration schemes until $W_K(\mathbf{r})$ and $\mu_K(\mathbf{r})$ converge.

$$W_K(\mathbf{r}) \longrightarrow W_K(\mathbf{r}) + \mathrm{constW} \times \left(W_K^{\rm target}(\mathbf{r}) - W_K(\mathbf{r})\right) \tag{2.109}$$

and

$$\mu_K(\mathbf{r}) \longrightarrow \mu_K(\mathbf{r}) + \mathrm{constV} \times \left(\phi_K^{\rm target}(\mathbf{r}) - \phi_K(\mathbf{r})\right), \tag{2.110}$$

where $W_K^{\rm target}$ is obtained by substituting $\phi_K^{\rm target}(\mathbf{r})$ into eq. (2.2). The $\phi_K(\mathbf{r})$ in eq. (2.110) is obtained using eq. (2.101) and the path integral $Q_K(s,\mathbf{r})$ calculated using eq. (2.96) and the self consistent fieled $V_K(\mathbf{r}) = W_K(\mathbf{r}) - \mu_K(\mathbf{r})$ in the previous iteration step. This iteration procedure is repeated until $V_K(\mathbf{r})$ and $W_K(\mathbf{r})$ converge within a certain error level. This gives the chemical potential $\mu_K(\mathbf{r})$ under the condition that the segment density is constrained to $\phi_K^{\rm target}(\mathbf{r})$. The criterion for the convergence of the above iteration scheme for the "equilibration under the constraint on the segment density" is slightly different from that for the "static equilibrium" calculation. In the present case, the iteration is finished when the maximum value of the amounts of updates for $W_K(\mathbf{r})$ and $\phi_K(\mathbf{r})$ during one iteration step becomes less than the error level. Then, $\mu_K(\mathbf{r})$ can be evaluated within the same error level. The error level should be specified in the input parameter list for SUSHI. A use of the relative error in $\phi_K(\mathbf{r})$ can also be specified in the judgement of the convergence.

2.9.11 Free energy

When the static calculation is converged or when each time step in the dynamic calculation is performed, the free energy can be evaluated using the formula,

$$\begin{aligned}\mathcal{F}\left[\{\phi_K\},\{V_K\}\right] &= -k_{\rm B}T\sum_p M_p \ln \mathcal{Z}_p + \mathcal{W}\left[\{\phi_K\}\right] \\ &\quad -\sum_K \int d\mathbf{r} V_K(\mathbf{r})\phi_K(\mathbf{r}) \\ &\quad +k_{\rm B}T\sum_p M_p \ln M_p\,. \end{aligned} \tag{2.111}$$

When the segment density in the bulk reservoir is given in the calculation using the grand canonical ensemble, the excess free energy can be evaluated using the following formula.

$$\mathcal{F}_{\text{exess}} = \mathcal{F} - \mathcal{F}^{(\text{bulk})} - \sum_p \mu_p (M_p - M_p^{(\text{bulk})}), \tag{2.112}$$

where $\mathcal{F}^{(\text{bulk})}$ is the free energy for a uniform system with the same composition as the bulk system, $M_p^{(\text{bulk})}$ and μ_p are the total numbers and the chemical potential of the p-type chain, respectively.

2.9.12 Efficient calculation method for polydisperse systems

SUSHI can simulate a polydisperse system by modeling it as a mixture of polymers with slightly different polymerization indices and with appropriately chosen volume fractions that correspond to the molecular weight distribution of the target polydisperse system. In SUSHI, however, the required memory and the CPU time are both increased as the total number of polymer components is increased. This will cause a serious problem when we try to simulate the polydisperse system using the above-mentioned modeling. However, for the static calculations on a system whose polydisperse components are only homopolymers, one can use a single path integral for all the homopolymers of the same kind but with different chain lengths. This allows us to avoid the above-mentioned difficulty. In the input UDF file for SUSHI, one can direct SUSHI to share the path integral of the longest chain among the group of polydisperse chains of the same kind. By using this option, SUSHI can simulate polydisperse homopolymer systems with practically infinite number of components, except for the difficulty in preparing the input data of the chain length and the volume fraction for each component. (In the future, we plan to offer a tool for preparing the input UDF file for the polydisperse systems using a few parameters such as M_w, M_n, etc.) The same technique can also be used for the static calculation of a mixture of A-B type diblock copolymers with different values of the block ratio. In this case, the path integrals starting from the free ends can be shared. For more details, refer to Chapter 5.

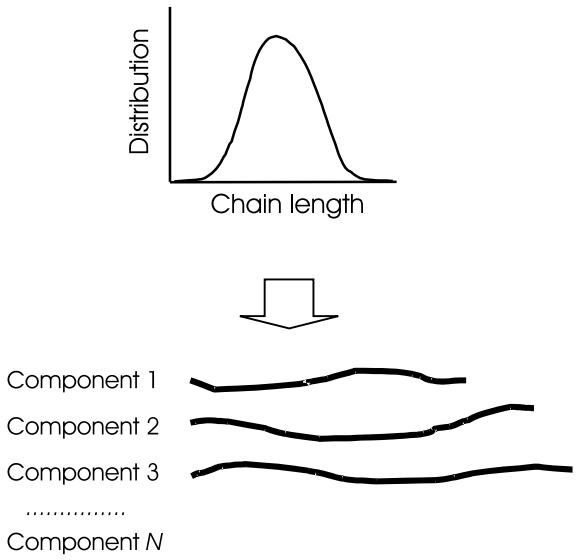


Figure 2.12: Schematic explanation of the model of a polydisperse polymer system used in SUSHI

2.9.13 Domain specification

Simply using the static calculation of SUSHI, it is usually difficult to obtain the true equilibrium state that corresponds to the global minimum of the free energy. Instead, the calculation is often trapped by one of the metastable states. This is due to the fact that the free energy has a huge number of local metastable minima. Therefore, a special technique is required to obtain the true equilibrium state with the desired domain morphology. Let us assume that we try to reproduce the well-known phase diagram of a diblock copolymer. In order to obtain the desired domain morphology as the final result of the static SCF calculation,

one should adjust the initial distribution of the self consistent field. Such a setup is also available in SUSHI. Readers should refer Chapter 5 for the details.

2.9.14 Mask operation to constrain junctions within regions

To simulate self-assembled structures of dilute polymeric components in a mixture, such as the micellar formation in a dilute block copolymer solution, one has to prevent the polymers from getting out of the micelle and from dissolving into the solvent. In SUSHI, this is realized by constraining the positions of specified junctions (or free ends) within a specified region. Readers should refer Chapter 5 for the details.

2.9.15 Dynamic mean field simulation on chemical reaction processes

It is possible to perform dynamical mean field simulations on several simple chemical reactions by adding a model reaction term to the right-hand-side of eq. (2.49). The types of reactions that can be simulated in SUSHI will be described in the following subsections. Readers should refer Chapter 5 for the details of how to prepare the input file for SUSHI.

Fast reactions

In case that the reaction rates are much faster than the diffusion process of the polymers, such as the radical polymerization reactions, it is reasonable to assume that the monomers instantaneously change into polymers with a certain degree of polymerization. Such a process is described by the following model equations:

$$\frac{\partial \phi^{(S)}(\mathbf{r},t)}{\partial t} = \frac{\partial \phi^{(S)}(\mathbf{r},t)}{\partial t}\Big|_{diffusion} - k\phi^{(S)}(\mathbf{r},t) \tag{2.113}$$

$$\frac{\partial \phi^{(P)}(\mathbf{r},t)}{\partial t} = \frac{\partial \phi^{(P)}(\mathbf{r},t)}{\partial t}\Big|_{diffusion} + k\phi^{(S)}(\mathbf{r},t). \tag{2.114}$$

The first term on the right-hand side is the contribution from the diffusion, which is the same as the right-hand side of eq. (2.49). The second term on the right-hand side is the reaction term, where k is a model reaction constant for the polymerization reaction, and $\phi^{(S)}(\mathbf{r},t)$ and $\phi^{(P)}(\mathbf{r},t)$ are the volume fractions of monomers and the polymers, respectively.

Reactions with active monomers or with active sites of polymers

If the reaction rate is of the same order as the diffusion rate of the polymers, one can consider the second order reaction process induced by reactive sites of the polymers to produce different types of polymers. One such example is the production process of an A-B diblock copolymer chain from a pair of an A-homopolymer and a B-homopolymer with reactive site on one of the chain ends. (See the figure below.)

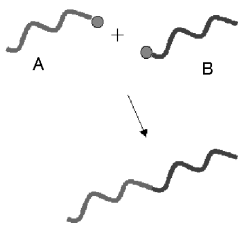


Figure 2.13: Schematic figure of the production of a diblock copolymer chain by a reaction at the reactive sites on the chain ends

Here, we consider only the canonical ensemble. The spatial distributions of these reactive sites are denoted as $\varphi^{(A)}_{reactive}(\mathbf{r},t)$ and $\varphi^{(B)}_{reactive}(\mathbf{r},t)$. The second order reaction of these reactive sites can be described as

$$\frac{\partial \varphi^{(A)}_{reactive}(\mathbf{r},t)}{\partial t} = -k\varphi^{(A)}_{reactive}(\mathbf{r},t)\varphi^{(B)}_{reactive}(\mathbf{r},t) \tag{2.115}$$

$$\frac{\partial \varphi^{(B)}_{reactive}(\mathbf{r},t)}{\partial t} = -k\varphi^{(A)}_{reactive}(\mathbf{r},t)\varphi^{(B)}_{reactive}(\mathbf{r},t)\ , \tag{2.116}$$

where k is the reaction constant. Let us denote the volume fractions of homopolymers consumed in this reaction per unit time as $\phi^{(A)}_{reactive}|_k$ and $\phi^{(B)}_{reactive}|_k$ respectively, whose explicit expression will be described later. These quantities mean the volume fractions of the homopolymers that are connected to the reacted site. Such amounts of the volume fractions should be subtracted from the distributions of the A-homopolymers and the B-homopolymers. At the same time, the same amount should be added to the distributions of the A-subchains and the B-subchains of the block copolymer, respectively. Then, the model evolution equations are written as follows.

$$\frac{\partial \phi^{(A)}_{reactant}(\mathbf{r},t)}{\partial t} = \frac{\partial \phi^{(A)}_{reactant}(\mathbf{r},t)}{\partial t}\Big|_{diffusion} - \phi^{(A)}_{reactive}(\mathbf{r},t)|_k \tag{2.117}$$

$$\frac{\partial \phi^{(B)}_{reactant}(\mathbf{r},t)}{\partial t} = \frac{\partial \phi^{(B)}_{reactant}(\mathbf{r},t)}{\partial t}\Big|_{diffusion} - \phi^{(B)}_{reactive}(\mathbf{r},t)|_k \tag{2.118}$$

$$\frac{\partial \phi^{(A)}_{product}(\mathbf{r},t)}{\partial t} = \frac{\partial \phi^{(A)}_{product}(\mathbf{r},t)}{\partial t}\Big|_{diffusion} + \phi^{(A)}_{reactive}(\mathbf{r},t)|_k \tag{2.119}$$

$$\frac{\partial \phi^{(B)}_{product}(\mathbf{r},t)}{\partial t} = \frac{\partial \phi^{(B)}_{product}(\mathbf{r},t)}{\partial t}\Big|_{diffusion} + \phi^{(B)}_{reactive}(\mathbf{r},t)|_k\ , \tag{2.120}$$

where $\phi^{(A)}_{reactant}(\mathbf{r},t)$ and $\phi^{(B)}_{reactant}(\mathbf{r},t)$ are the distributions of the A- and B-homopolymers (reactants), $\phi^{(A)}_{product}(\mathbf{r},t)$ and $\phi^{(B)}_{product}(\mathbf{r},t)$ are the distributions of the A- and B-subchains of the block copolymers (products). The first term on the right-hand side is the contribution from the diffusion similar to that introduced in the proceeding paragraph. Although this example is simple, we can consider other reaction processes that involve polymers with more complex structures. In such a case, one has to solve a set of time evolution equations for each of the subchains of all polymers.

In the actual SCF calculations, as was described in fig. 2.2 in Section 2.2, all the free ends of the chains and all the connected points between subchains are treated as "junctions". These junctions can be considered as the reactive sites. Once the convergence of the SCF iteration is achieved, the spatial distributions of the reactive sites are obtained as follows:

$$\varphi^{(A)}_{reactive}(\mathbf{r},t) = C^{(A)}_r \prod_r \widetilde{Q}_K(N^{(A)}_r,\mathbf{r}) \tag{2.121}$$

$$\varphi^{(B)}_{reactive}(\mathbf{r},t) = C^{(B)}_r \prod_r \widetilde{Q}_K(N^{(B)}_r,\mathbf{r})\ , \tag{2.122}$$

where $\prod_r$ means product of all the final values of the path integrals coming into the reactive site. Since the canonical ensemble is considered, the normalization constants $C^{(A)}_r$ and $C^{(B)}_r$ given by eq. (2.19) are independent of the index of the sub-chain r. Let us consider the total numbers of A- and B-homopolymers consumed in the reaction per unit time and denote them as $M^{(A)}_{reactive}$ and $M^{(B)}_{reactive}$. These values are obtained by multiplying the number of polymers before the reaction by the reactive fraction of the reactive sites as

$$M^{(A)}_{reactive} = M^{(A)}\frac{\int d\mathbf{r}k\varphi^{(A)}_{reactive}(\mathbf{r},t)\varphi^{(B)}_{reactive}(\mathbf{r},t)}{\int d\mathbf{r}\varphi^{(A)}_{reactive}(\mathbf{r},t)} \tag{2.123}$$

$$M^{(B)}_{reactive} = M^{(B)}\frac{\int d\mathbf{r}k\varphi^{(A)}_{reactive}(\mathbf{r},t)\varphi^{(B)}_{reactive}(\mathbf{r},t)}{\int d\mathbf{r}\varphi^{(B)}_{reactive}(\mathbf{r},t)}\ , \tag{2.124}$$

where $M^{(A)}$ and $M^{(B)}$ are the numbers of polymers in the system before the reaction. Finally, by using eqs. (2.101) and (2.102) and the converged self consistent field, $\phi^{(A)}_{reactive}(\mathbf{r},t)|_k$ and $\phi^{(B)}_{reactive}(\mathbf{r},t)|_k$ are obtained as follows.

$$\phi^{(A)}_{reactive}(\mathbf{r},t)|_k = C^{(A)}_{reactive}\sum_i \int d\mathbf{r}_0 \int d\mathbf{r}_N k\varphi^{(B)}_{reactive}(\mathbf{r}_0,t)q_0(\mathbf{r}_0)Q_A(0,\mathbf{r}_0;i,\mathbf{r})Q_A(i,\mathbf{r};N,\mathbf{r}_N)q_N(\mathbf{r}_N) \tag{2.125}$$

$$C^{(A)}_{reactive} = \frac{M^{(A)}{}_{reactive}}{\int d\mathbf{r}_0 \int d\mathbf{r}_N q_0(\mathbf{r}_0) Q_A(0,\mathbf{r}_0;N,\mathbf{r}_N) q_N(\mathbf{r}_N)} \tag{2.126}$$

$$\phi^{(B)}_{reactive}(\mathbf{r},t)|_k = C^{(B)}_{reactive} \sum_i \int d\mathbf{r}_0 \int d\mathbf{r}_N k \varphi^{(A)}_{reactive}(\mathbf{r}_0,t) q_0(\mathbf{r}_0) Q_B(0,\mathbf{r}_0;i,\mathbf{r}) Q_B(i,\mathbf{r};N,\mathbf{r}_N) q_N(\mathbf{r}_N) \tag{2.127}$$

$$C^{(B)}_{reactive} = \frac{M^{(B)}{}_{reactive}}{\int d\mathbf{r}_0 \int d\mathbf{r}_N q_0(\mathbf{r}_0) Q_B(0,\mathbf{r}_0;N,\mathbf{r}_N) q_N(\mathbf{r}_N)} \,, \tag{2.128}$$

where $q_0(\mathbf{r}_0)$ and $q_N(\mathbf{r}_N)$ are the statistical weights of the reactive end ($i = 0$) and the other non-reactive end ($i = N$). In the present example, as the reactants are homopolymers, the value of both of $q_0(\mathbf{r}_0)$ and $q_N(\mathbf{r}_N)$ are unity because they are free ends. These values take non-trivial values ($\neq 1$) only when the reactant polymer is composed of multiple subchains.

Grafting reaction

By using the procedure described in the previous section, it is possible to simulate a graft reaction where free chains are grafted to solid walls. Let us assume that A-homopolymer is grafted at the reactive end to a solid wall from a polymer solution or from a melt. This graft reaction can be regarded as a kind of chemical reaction where A-homopolymer changes to grafted G-homopolymer. We denote the densities of the reactive sites of A and G polymers at the wall position as $\varphi^{(A)}_{reactive}(\mathbf{r}_w,t)$ and $\varphi^{(G)}_{reactive}(\mathbf{r}_w,t)$, where $\mathbf{r}_w$ denotes the position vector of a point on the wall. The reaction equations for the reactive sites are as follows:

$$\frac{\partial \varphi^{(A)}_{reactive}(\mathbf{r}_w,t)}{\partial t} = -k\varphi^{(A)}_{reactive}(\mathbf{r}_w,t) \tag{2.129}$$

$$\frac{\partial \varphi^{(G)}_{reactive}(\mathbf{r}_w,t)}{\partial t} = k\varphi^{(A)}_{reactive}(\mathbf{r}_w,t) \,, \tag{2.130}$$

where k is the reaction constant. The model equations for this graft reaction are expressed as

$$\frac{\partial \phi^{(A)}(\mathbf{r},t)}{\partial t} = \frac{\partial \phi^{(A)}(\mathbf{r},t)}{\partial t}\Big|_{diffusion} - \phi^{(A)}_{reactive}(\mathbf{r},t)|_k \tag{2.131}$$

$$\frac{\partial \phi^{(G)}(\mathbf{r},t)}{\partial t} = \frac{\partial \phi^{(G)}(\mathbf{r},t)}{\partial t}\Big|_{diffusion} + \phi^{(A)}_{reactive}(\mathbf{r},t)|_k \,, \tag{2.132}$$

where $\phi^{(A)}(\mathbf{r},t)$ and $\phi^{(B)}(\mathbf{r},t)$ are the volume fractions of A- and G-homopolymers, and $\phi^{(A)}_{reactive}(\mathbf{r},t)|_k$ is the volume fraction of the A-homopolymer consumed in the reaction per unit time which can be calculated by the similar manner described in the previous section. The volume fraction of the reactive site is calculated as

$$\varphi^{(A)}_{reactive}(\mathbf{r}_w,t) = C^{(A)}_r \prod_r \widetilde{Q}_K(N^{(A)}_r,\mathbf{r}_w) \,, \tag{2.133}$$

where $C^{(A)}_r$ is defined in eq. (2.19). Then, the number of A-homopolymers consumed in the reaction per unit time is given by

$$M^{(A)}_{reactive} = M^{(A)} \frac{\int d\mathbf{r} k\delta(\mathbf{r}-\mathbf{r}_w)\varphi^{(A)}_{reactive}(\mathbf{r}_w,t)}{\int d\mathbf{r}\varphi^{(A)}_{reactive}(\mathbf{r}_w,t)}. \tag{2.134}$$

The volume fraction of A-homopolymer consumed in the reaction per unit time is calculated by

$$\phi^{(A)}_{reactive}(\mathbf{r},t)|_k = C^{(A)}_{reactive} \sum_i \int d\mathbf{r}_0 \int d\mathbf{r}_N k\delta(\mathbf{r}-\mathbf{r}_w) q_0(\mathbf{r}_0) Q_A(0,\mathbf{r}_0;i,\mathbf{r}) Q_A(i,\mathbf{r};N,\mathbf{r}_N) q_N(\mathbf{r}_N) \,, \tag{2.135}$$

where $C^{(A)}_{reactive}$ is give by eq. (2.126), and $q_0(\mathbf{r}_0)$ and $q_N(\mathbf{r}_N)$ are statistical weights of both ends of the chain, which are both unity for the present homopolymer case.

2.9.16 Strong polyelectrolyte

Strong polyelectrolyte is a polymer solution in which almost all the ionizable atomic groups are ionized. Therefore, the charges on the chain can be treated as constant. Then, the strong polyelectrolyte can be simulated by assigning permanent electric charge to each segment and by calculating the electrostatic potential of the system. The electrostatic potential is treated as an external field acting on the segments and is added to the self-consistent field. We denote the dielectric constant of the system as ϵ_0 and the specific dielectric constant of K-type segment as ϵ_K then assume the local dielectric constant as

$$\epsilon(\mathbf{r}) = \epsilon_0 \sum_K \epsilon_K \phi_K(\mathbf{r}). \tag{2.136}$$

We also denote the electric charge of the K-type segments per unit volume as ρ_K and the distribution of the total charge in the system is written as

$$\rho(\mathbf{r}) = \sum_K \rho_K \phi_K(\mathbf{r}). \tag{2.137}$$

The electrostatic potential $U(\mathbf{r})$ is obtained by solving the following Poisson equation.

$$\nabla\epsilon(\mathbf{r})\nabla U(\mathbf{r}) = -\rho(\mathbf{r}). \tag{2.138}$$

The electrostatic potential energy $\mathcal{W}_e$ is calculated by

$$\mathcal{W}_e = \frac{1}{2}\int d\mathbf{r}\int d\mathbf{r}'\,\frac{\rho(\mathbf{r})\rho(\mathbf{r}')}{4\pi\epsilon|\mathbf{r}-\mathbf{r}'|} = \frac{1}{2}\int d\mathbf{r}U(\mathbf{r})\rho(\mathbf{r}). \tag{2.139}$$

The external potential per a K-type segment due to the electrostatic interaction is given by $U(\mathbf{r})\rho_K$. The effect of the electric charge is taken into the SCF calculation simply by adding the potential $U(\mathbf{r})\rho_K$ to the self-consistent field defined by eqs. (2.1).

2.9.17 SCF Monte Carlo method

One of the limitations of the SCF method is the difficulty in simulating dilute polymer solutions. This is because the fluctuations in the density distribution of the polymers in dilute solutions are so large that the mean field approximation used in the SCF theory cannot be justified. As a result, the polymer density calculated by the standard SCF method spreads uniformly over the whole system. To treat an isolated chain in a dilute solution, some tricks should be introduced. One such method is the mask operation mentioned introduced in Section 2.9.14. Here, we propose a Monte Carlo calculation of an isolated chain. We evaluate the free energy of the chain by the SCF calculation fixing the positions of some specified junctions using the mask operation. Then, the positions of these specified junctions are moved using the Monte Carlo procedure. This SCF Monte Carlo algorithm is as follows.

1) Select the junctions to be masked and set their initial positions.
   Set the counter for the Monte Carlo step $i = 0$.

2) Perform the static equilibrium calculation of the SCF.
   Evaluate the free energy of the system A^i.

3) Select one of the junctions and move it to one of
   its nearest neighbor lattice points randomly..

4) Perform the static equilibrium calculation of the SCF.
   Evaluate the free energy of the system A^{i+1}.

5) Calculate the difference in the free energy dA by
   $dA = A^{i+1} - A^i$.

6) If $dA < 0.$, reserve the updated configuration $i + 1$.
   Otherwise, calculate the quantity $p = exp(-dA/k_BT)$.
   Then, generate a random number $rand$ that distributes
   uniformly between 0. and 1. and
   If $p > rand$, reserve the updated configuration $i + 1$, and otherwise restore the old configuration i.

7) Return to 3) and repeat.

2.9.18 System size optimization

To optimize the lenght of the edge of the system is important to determin the final structure of micro phase separation. We introduce the system size optimization method to minimize the free energy density by changing the lenght of the edge of the system. We denote the free energy of system as $\mathcal{F}$,the volume of system as V and the method is given by the equation

$$\frac{\partial(\mathcal{F}/V)}{\partial X_i} = 0 \tag{2.140}$$

where X_i $(i = x, y, z)$ is the length of the edge of the system. The differential term of the right hand side can be solved numerically, so the equation can be solved as a numerical minimiztion problem.

In dynamic calculation, we can minimize the free energy density to couple the time dependent equation of segment (2.49) and the time dependent equation of the system size given by

$$\frac{\partial X_i}{\partial t} = -Q_i \frac{\partial(\mathcal{F}/V)}{\partial X_i} \tag{2.141}$$

where Q_i $(i = x, y, z)$ is a positive constant. This method is effective to get the global minimum structure of micro phase separation. Although this method does not conserve the system volume. To escape a big volume change, the introduction of elastic energy to the free energy is one solution, the equation of free enegy density $\mathcal{F}/V$ is extended as

$$\frac{\mathcal{F}_c}{V} = \frac{\mathcal{F}}{V} + \frac{1}{2\kappa}\frac{(V - V_0)^2}{V_0^2} \tag{2.142}$$

where κ is the compressibility of system and V_0 is the initial voluem of system.

2.9.19 Conclusion

In this chapter, we described how to perform numerical calculations based on the theoretical framework given in the previous chapter. As a summary, we clarify the mutual relations between the quantities appearing in the theory.

First, we summarize the self consistent field technique. The calculation scheme is shown in fig. 2.14. The basic task in the self consistent field theory is to obtain the segment density distribution, the external force (self consistent field) $V_K(\mathbf{r})$ acting on the segment density field, and the statistical weight ($Q_K(s,\mathbf{r})$), in a self consistent manner within the constraints imposed on the system. To realize this task, one first calculates the path integral $Q_K(s,\mathbf{r})$ at position $\mathbf{r}$ using eq. (2.7) with the self consistent filed $V_K(\mathbf{r})$. Then, the segment density $\phi_K(\mathbf{r})$ is calculated using $Q_K(s,\mathbf{r})$ through eq. (2.9). Finally the self consistent field $V_K(\mathbf{r})$ is obtained using $\phi_K(\mathbf{r})$ and eqs. (2.104) and (2.105). In the actual numerical calculation, using an appropriate initial values for $V_K(\mathbf{r})$, the above scheme is solved iteratively. This procedure is repeated until the change in $\phi_K(\mathbf{r})$ during the single iteration step becomes smaller than a threshold. When the change becomes small enough, the self consistent set of quantities is obtained.

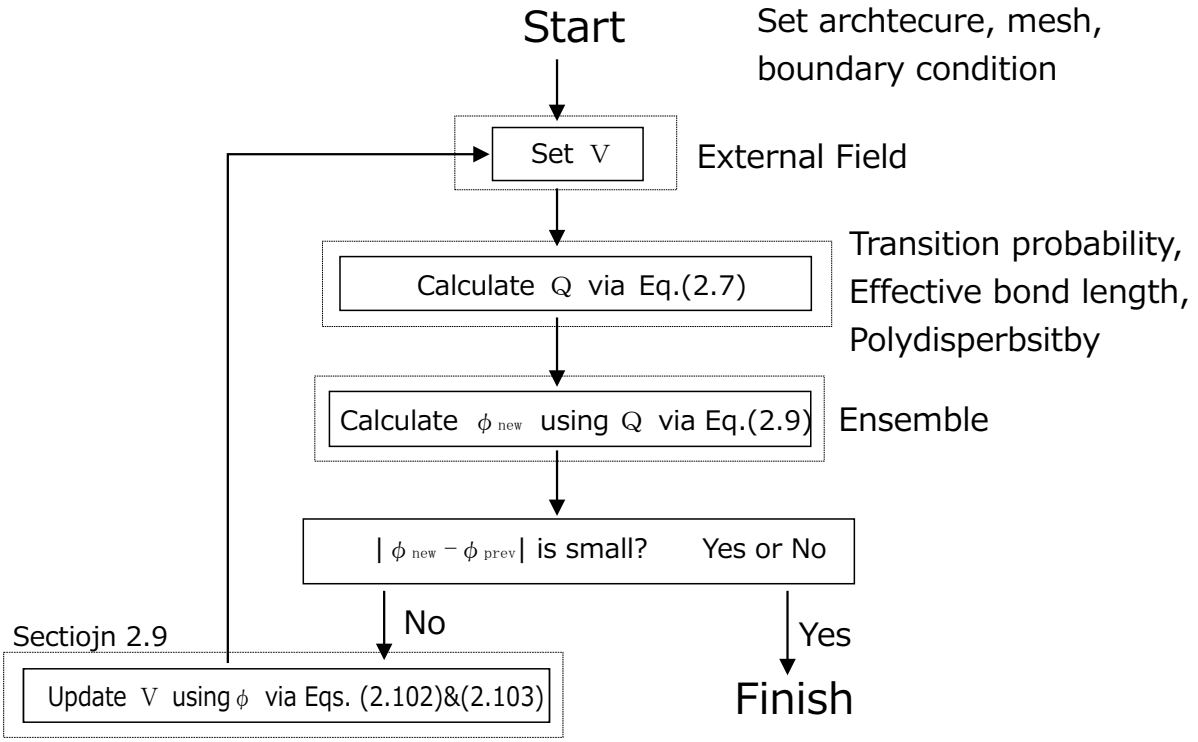


Figure 2.14: Flow chart of SCF calculation

By solving the self consistent set of equations, one can study various polymer systems. Appropriately choosing the polymerization processes or the catalysts, one can synthesize polymers with various branch structures or loop structures. To obtain path integrals for these complex polymers, one has to choose suitable techniques taking the chain structures into account. In Section 2.9.1, we have already discussed how to model various chain architectures in the actual computation. The spatial discretization methods, i.e. the spatial meshes and the boundary conditions, were discussed in Sections 2.9.3 and 2.9.4. An extended computational method of the $Q_K(s,\mathbf{r})$ that has an applicability to polymer chains with complicated internal structures, for example, the tapered polymers, etc. were also given. In this method, we introduced internal states for each subchain of the polymer chain. To calculate the path integral for such subchains with internal states, we introduced a transition probability between the internal states (in Section 2.3). We also introduced another model parameter, the effective bond length, to calculate the path integrals. Two statistical ensembles, i.e. the canonical ensemble and the grand canonical ensemble, were used to specify the equilibrium condition of the system used in the calculation of the path integrals and the density fields. (See Sections 2.2 and 2.9.8.) Other important properties of polymers, such as the grafting, the adsorption and the polydispersity, are also discussed in Sections 2.9.6, 2.9.7 and 2.9.12, respectively.

The dynamic mean field method is a modified version of the self consistent field theory, where the dynamical change of the density fields are allowed. In this scheme, the density fields are first given. Then, $V_K(\mathbf{r})$ and $Q_K(s, \mathbf{r})$ are determined using the density fields. In this dynamic case, we have to add another iteration scheme for the time evolution to the flow chart for the static SCF calculation shown in fig. 2.14. As a result, the flow chart for the dynamic mean field simulation is given in fig. 2.15.

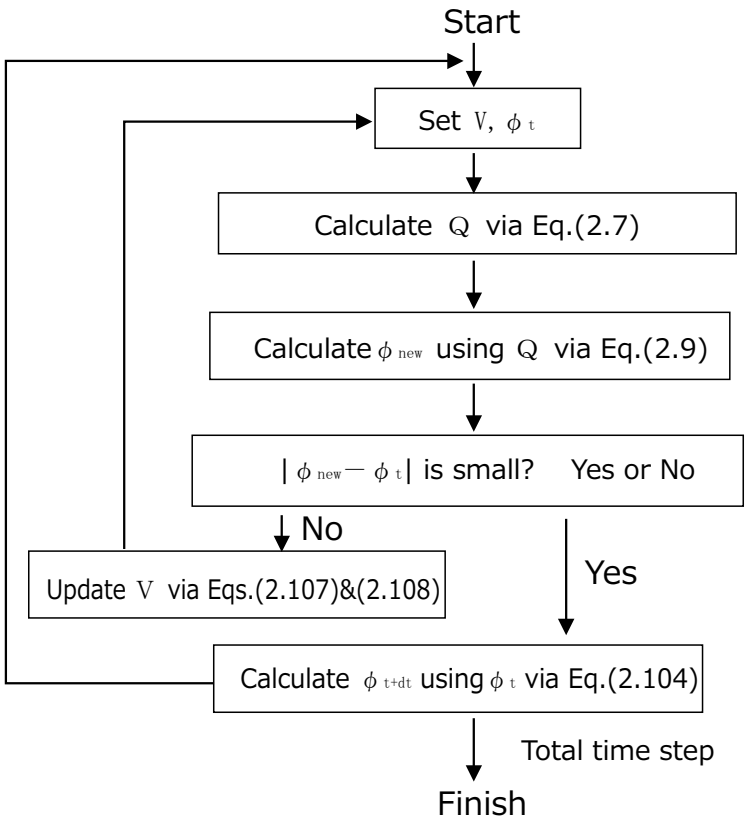


Figure 2.15: Flow chart of the dynamic mean field method

The peculiar steps of the dynamic calculation will be those described by eqs. (2.109) and (2.110), i.e. the updating $V_K(\mathbf{r})$ and other quantities.

Chapter 3

Starting SUSHI

3.1 Notes

SUSHI can be started by typing a command from the console. The supported operating systems are MS Windows and Linux. Although SUSHI has no implementation of GUI, one can use SUSHI as if it is running on a GUI by using the UDF format files on GOURMET.

The current version of SUSHI can read/write files with two different formats, i.e. the UDF format and the SEED format. Although the UDF format appears to be rather redundant, it has a well-defined structure and therefore prevents us from making mistakes when preparing the input files. Please refer to UDF manual about the details of the UDF format. On the other hand, the SEED format has a simple structure. Thus, it is easy for the user to prepare the SEED files although he/she has to type the keywords by himself/herself. The file of SEED format should obey the UNIX-type format (each line ends with LF).

In Chapter 5, one can find how to start SUSHI.

In order to prepare an input file for SUSHI, the easiest way will be to make a copy of a sample file and to modify it. The UDF format files can easily be edited on GOURMET. Editing with a text editor is also possible.

3.2 Examples of 1-dimensional calculation

Using the sample files contained under the directory SUSHI/sample, let us see how to prepare the input files for the SCF calculation. We start with the example of the static equilibrium calculation for one-dimensional systems.

3.2.1 Interfaces

As a simple example, let us consider an interface of an A/B polymer blend in one dimension. As shown in the following figure, the system is a phase separating A/B polymer blend. Both ends of the system are assumed to be reflective walls, and an interface is placed at the center.

Figure 3.1: A schematic picture of an interface in an A/B polymer blend

Let us start GOURMET and open the input file named interface_uin.udf and select the “Tree” button in the “View” box. Open the SUSHIInput folder in the left-side window to see its contents. You can see many folder icons. Each folder icon indicates a data structure. Hereafter, we call such a folder icon as “sub-folder”. The first sub-folder is ”calculation_method”. Open this sub-folder and you can see the selectable item “type” whose value is now specified as STATICS as is shown in the right-side window. The second item of the SUSHIInput is “start_condition”, which is specified as the select type with the value “START”. Next, open the ”solver_parameter” sub-folder. The type in this sub-folder is set as SCF.

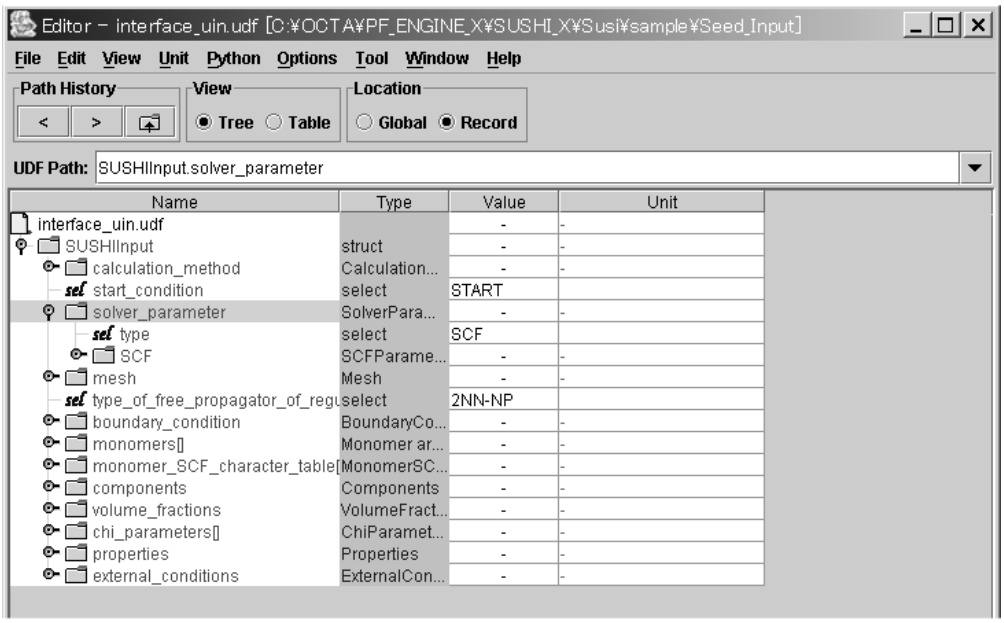


Figure 3.2: GOURMET initial window

The SUSHIInput folder has so many sub-folders. If the SUSHI would not start until we specify all the parameters contained in these sub-folders, nobody would dare to use it! Don’t worry. In order to start SUSHI, you have to specify only a few parameters explicitly.

The necessary items to start calculations are

1) Controlling parameters for the SCF calculation

2) Spatial mesh

3) Components of the system and their compositions

4) χ-parameters for the interactions between segments

in order to obtain the interfacial structure quickly, you are advised to specify

5) External conditions, by which the regions of the A and B domains are specified.

Only these input data are required to start the simulation.

Hereafter, for the sake of the GOURMET users, we will discuss the detail of the UDF input file. If you know the concept of "structure" in the program language C, you will be able to understand the following descriptions easily. In the following, the structure of the variables and the corresponding actual values of the variables will be described simultaneously.

Let us see how the variables are specified in the UDF input file using the "interface_uin.udf" file as an example. Please open this UDF file with an editor or with a word processor. You can not use simple editors that can read/write only DOS format files. A high-class editor that can read/write UNIX format should be used instead. At the beginning of the file, there is the "definition part" where the data structures are declared. Various information on the data are described here for the sake of the users. Next part is the "data part", where the actual data are written.

```
\begin{data}
```

Please search for the keyword "`begin{data}`" using the search function of the editor/word processor. Below this keyword, you will find the actual data.

First step

As was explained at the beginning of this section, you have to specify two data. In this UDF file, these data are specified as follows,

---

```
// Data
\begin{data}
SUSHIInput:{ // SUSHIInput/
  calculation_method { //CalculationMethod
    type "STATICS"
    .....
  }
  start_condition "START"
  .....
```

---

where the static equilibrium calculation is chosen, and the restarting option is disabled. This data structure is defined as follows.

```
// Data definition
class CalculationMethod:{ // The data structure of calculation method.
    // The data type "select" means that user can select the string data from {...}.
    type:select { "STATICS", "DYNAMICS", "MONTECARLO" }
    DYNAMICS:DynamicsParameter
    MONTECARLO:MonteCarloParameter
        // The DynamicsParameter and the MonteCarloParameter are data structures
        // defined in other classes.
}
SUSHIInput:{
  // Calculation control data ##############################
    calculation_method:CalculationMethod
    start_condition:select
        // The flag for starting condition
        // START            : normal start
        // CONTINUE         : continue with reading the mesh at the final recoerd
        // RESTART          : restart without reading the mesh at the final recoerd
        // RESTART_READMESH : restart with reading the mesh at the final recoerd
    .....
}
```

Control parameters for the SCF calculation

Here, the solver parameters for SCF calculation are explained. The input data in the sample UDF file are as follows.

```
// Data
  solver_parameter { //SolverParameter
    type "SCF"
    SCF { //SCFParameter011031
      delta_s 1
      constV 0.05
      constW 0.1
      error 0.0001
      random_seed 0
      standard_deviation 0.00015
      judge_method "ABSOLUTE"
      convergence_test_interval_step 0
      max_SCF_step 20000
      scf_output_interval_step 0
      SCF_method "INCORE"
      pathintegral_scheme "EXPLICIT"
    }
    ....
  }
```

Although so many parameters are listed, only the following parameters are actually used in this case.

```
        delta_s:double                // The mesh width for the chain length used in the
                                      // calculation of the path integral.
                                      // This parameter is positive and the same value
                                      // is used in the calculations of the path
                                      // integral for both directions along the chain.
        constV:double                 // The constant parameter used in the updating of
                                      // the chemical potential in the SCF iteration
                                      // scheme.
        constW:double                 // The constant parameter used in the updating of
                                      // the interaction between segments in the SCF
                                      // iteration scheme.
        error:double                  // The threshold value used in the judgement of
                                      // the convergence of the iterations.
        standard_deviation:double     // The standard deviation of the Gaussian random
                                      // numbers used as the initial values of the
                                      // segment density fields.
        max_SCF_step:int              // The maximum number of iterations allowed for
                                      // the SCF calculation.
```

The syntax of these structured data is as follows.

```
// Data definition
class SolverParameter:{ // The ‘‘solver" means a specialized simulator.
    type:select { "ADF", "FH", "SCF" }
```

```
    SCF:SCFParameter // The SCF (Self Consistent Field) method.
    ADF:ADFParameter // The ADF (Approximate Density Functional) method.
                     // (Under construction)
    FH:FHParameter   // The Cahn-Hilliard type dynamics method.
                     // (Under construction)
}
class SCFParameter:{
        delta_s:double                 // The mesh width for the chain length used in the
                                       // calculation of the path integral.
                                       // This parameter is positive and the same value
                                       // is used in the calculations of the path
                                       // integral for both directions along the chain.
        constV:double                  // The constant parameter used in the updating of
                                       // the chemical potential in the SCF iteration
                                       // scheme.
        constW:double                  // The constant parameter used in the updating of
                                       // the interaction between segments in the CF
                                       // iteration scheme.
        error:double                   // The threshold value used in the judgement of
                                       // the convergence of the iterations.
        random_seed:int                // A seed for the random number generator.
        standard_deviation:double      // The standard deviation of the Gaussian random
                                       // numbers used as the initial values of the
                                       // segment density fields.
        method_of_convergence_test:select { "ABSOLUTE", "RELATIVE" }
                                       // The method to judge the convergence of the
                                       // iterations. This parameter is used only in the
                                       // dynamic SCF calculations.
        convergence_test_interval_step:int  // Interval step of the above test.
                                       // Default 0 that is recognized 1. i.e. every step.
        max_SCF_step:int               // The maximum number of iterations allowed for
                                       // the SCF calculation.
        scf_output_interval_step:int   // During the SCF iterations, the data are written
                                       // into the file/screen at every other
                                       // "output_interval_step" iteration steps.
        SCF_method:select { "INCORE", "DIRECT" }
                                       // How to handle the memory area for the large
                                       // data of the path integrals.
        pathintegral_scheme:select { "EXPLICIT", "IMPLICIT" }
                                       // The type of the scheme to integrate the
                                       // evolution equation for the path integral.
}
```

When you use the regular mesh and assume that both the mesh width and the effective bond length b are unity and the χN value is not so large, it is appropriate to use the value 1 for delta_s.

The two constant parameters constV and constW are magic numbers, which strongly affect the speed of the convergence of the SCF iterations. You have to adjust these parameters depending on the conditions of the system you are simulating. One of the most difficult operations in using SUSHI would be how to select appropriate values for these parameters. However, after using SUSHI several times, you will be already an expert of SUSHI and can select suitable values for these parameters (Wonderful!). If you are not yet familiar with SUSHI, you are kindly advised to remember that constV = 0.05 and constW = 0.1 will usually be the maximum values in the static equilibrium calculations. Increasing the χ parameter values (i.e. increasing the repulsive interaction between segments) will lead to a slower convergence of the SCF iterations, and sometimes the SCF iterations will never converge. In such a case, make both contV and constW smaller. In the example, the ratio between these parameters is taken to be 5:10. However, you do not need to keep this

ratio. You are free to change these two parameters independently so that the iteration scheme will converge. If you choose too small values for these parameters, the total number of SCF iterations will be enormously increased. If the convergence is too slow, it may be a good choice to slightly increase the values of these parameters.

There are certain cases where the SCF iterations will not converge whatever you choose values for the parameters. In such a case, how many SCF iteration steps should we wait? Empirically, you should abandon the calculation if you cannot see a sign of convergence after 20,000 SCF iterations. In the provided sample files, we present many examples of input parameters with which convergence of the SCF iteration scheme has been confirmed. These input files can be used as references for your simulation runs.

The parameter "error" is used as the threshold value for the judgement of the convergence of the volume fractions during the SCF iterations. For test calculations, error = 0.0001 will be enough. The parameter "standard_deviation" specifies the standard deviation of the Gaussian random noise given as the initial values of the self consistent fields for the static equilibrium calculation. We recommend you to choose this as about 1.5 times as the value of "error".

Do not forget to specify the parameter "max_SCF_step" that limits the maximum number of the SCF iterations.

Mesh

Let us see how to specify the spatial mesh. The syntax of the UDF data "SUSHIInput.mesh" is as follows.

```
// Data
  mesh { // Mesh
    name "test"
    type "REGULAR"
    axes [
      id0 { // MeshAxis
        values [ 0 32 32 ]
      }
    ]
    index_rule [ 0 1 2 ]
  }
  type_of_free_propagator_of_regular_mesh "2NN-NP"
```

The detailed definitions of this structured data are as follows.

```
// Data definition
class MeshAxis:{
    values[]:double
        // In case of the regular, spherical, or cylindrical mesh,
        // this array contains the minimum and the maximum values
        // of the range along the axis, and the total number of
        // divided cells along the axis, respectively.
        // In case of the rectangular mesh, an array of the coordinates
        // of the mesh points are stored.
}
class Mesh:{
    name:KEY // The data type "KEY" is a character string, which is used to search the
             // target data structure.
    type:select { "REGULAR", "RECTANGULAR", "CYLINDRICAL", "SPHERICAL" }
    axes[]:MeshAxis
        // Array of the mesh axes.
        // Its dimension is the total number of coordinate axes of the system.
    index_rule[]:int
        // This specifies how the array elements (i, j, k) are arranged on the memory.
```

```
        // The first argument i runs first for example (0,0,0),(1,0,0)...(X-1,Y-1,Z-1).
        // For SUSHI, this is fixed as [0, 1, 2] except cylindrical mesh and
        // is fixed as [2, 1, 0] for cylindrical mesh.
        // Although this parameter does not affect the functions of SUSHI,
        // it is used when the mesh data is passed to another simulator and
        // is passed to the viewer on GOURMET.
}
SUSHIInput:{
    ....
    mesh:Mesh
    type_of_free_propagator_of_regular_mesh:select { "1NN-P", "2NN-NP", "2NN-P", "3NN-P" }
        // The type of the discretized Laplacian operator.
        // This is for the regular mesh only.
    ....
}
```

In the sample file, the name of the mesh is specified as "test", and it is declared as a "regular mesh". In the "MeshAxis" parameter, only a single data "id0" is specified. Since the dimension of "MeshAxis" corresponds to the space dimension in the regular mesh case, we recognize that this system is one dimensional. Three values are stored in this parameter. These values mean the minimum and the maximum values of the range along the axis, and the number of mesh celles in it. In the example, the X-axis starts from 0 and finishes at 32, and is divided into 32 cells. If you want to simulate a multidimensional system, you have only to add extra elements to the array "MeshAxis". Please refer to the GOURMET manual for how to add extra data. For SUSHI, the parameter "index_rule" is fixed as [0, 1, 2]. Although SUSHI can run without this parameter, absence of this parameter may cause unpredictable behavior when the data is passed to another simulator. Please do not forget to specify this parameter as $[0, 1, 2]$.

In the GOURMET window, you can see the following.

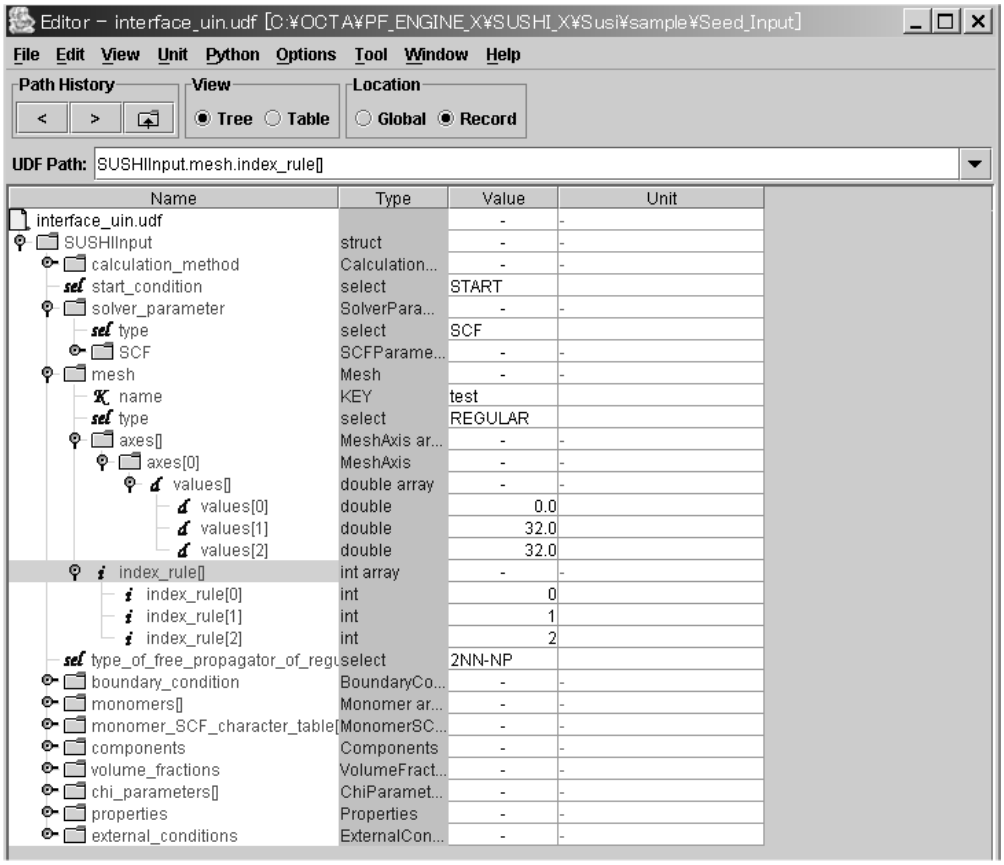


Figure 3.3: Mesh data window

A mesh should be supplemented with the boundary conditions. The data structure of UDF "SUSHIInput.boundary_condition" are as follows.

```
// Data
  boundary_condition { // BoundaryCondition
    conditions [
      id0 { // AxisBoundaryCondition
        axis_conditions [ "NEUMANN"  "NEUMANN"  ]
      }
    ]
    .....
  }
// Data definition
class AxisBoundaryCondition:{ // The boundary conditions at the both ends of the axis
                              // are specified.
    axis_conditions[]:string
        // PERIODIC :Periodic boundary conditions
        //          In this case, specifying only axis_condition[0] is enough.
        // DIRICHLET or WALL :Absorbing wall,
        //    i.e. the Dirichlet boundary condition with the boundary value 0.
        // NEUMANN :Reflective wall,
        //    i.e. the Neumann boundary conditions with vanishing gradient of the field.
}
class BoundaryCondition:{
    conditions[]:AxisBoundaryCondition // An array of the boundary conditions for each
                                       // axis.
```

```
    ....
}
SUSHIInput:{
    .....
    boundary_condition:BoundaryCondition
        // Boundary conditions.
        // Refer to Section 2.7.4.
    .....
}
```

In the sample UDF file, the both ends of the axis of the mesh are specified as reflective boundaries. If you want to specify them as solid walls, set WALL or DIRICHLET to axis_conditions. Now, the specification of the mesh has been completed.

Molecule

Let us specify the molecules contained in the system. A polymer is composed of monomers. In SUSHI, "monomer" means a group of several chemical repeating units of the polymer, and is identified with the segment of a Gaussian chain. First, let us see the input UDF data "SUSHIInput.monomers".

```
// Data
  monomers [
    id0 { // Monomer
      species_name "A"
      specific_volume 1
      effective_bond_length 1
    }
    id1 { // Monomer
      species_name "B"
      specific_volume 1
      effective_bond_length 1
    }
  ]
```

This data structure is simple. The parameters you have to specify are the names of the monomers, the specific volumes, and the effective bond lengths. In the sample file, the names of the monomers are specified as A and B, and the specific volumes and the effective bond lengths are all set to unity. Usually, you should specify only the parameter "species_name" leaving the other parameters as unity. The definition of the data structures of the "Monomer" and the "Solvent" are as follows.

```
// Data definition
class Monomer:{
    species_name:string           // Name of the monomer.
    specific_volume:double        // Specific volume of the monomer.
    effective_bond_length:double  // Effective bond length corresponding to a single
                                  // monomer.
}
class Solvent:{
    name:string                   // Name of the solvent.
    specific_volume:double        // Specific volume of the solvent.
}
```

The data structure of the "Solvent" is simpler than that of the "Monomer". It possesses only the name and the specific volume. How are actual polymer chains constructed using these monomers? The following is the

definition of polymers.

```
// Data definition
class Block:{         // Sub-chain.
    monomer_name:string
        // The name of the monomer which constitutes this sub-chain.
        // This monomer name must be defined as a Monomer in advance
    number_of_monomer:double
        // The total number of monomers contained in this sub-chain.
}
class JunctionPair:{  // A pair of the ID's of the junction points at the both ends of
                      // this sub-chain.
    first:int
    second:int
}
class Polymer:{
    type:select {"HOMO","BLOCK","COMB","STAR","GENERAL" }
        // The keyword specifying the branching structure of the chain,
        // i.e. how to connect the subchains.
        // HOMO :  A homo polymer.
        // BLOCK : A linear multiblock copolymer made of a sequence of the sub-chains
        //         with the same order as they are stored in this array.
        // STAR :  A star block copolymer made of sub-chains which are connected
        //         at a single junction point.
        // COOMB : A comb-type block copolymer.
             // The order of the sub-chains stored in the array is as follows.
             // main chain - side chain - main chain - side chain -.....
             // For example, an array of the sub-chains A1, B1, A2, B2, and A3
             // indicates the following structure.
             //     A1--+--A2--+--A3
             //         B1     B2
        //  GENERAL : The way how the sub-chains are connected to each other
             //       is specified by the ID's of the two junction points at the
             //       both ends of each sub-chain.
    blocks[]:Block                // Array of sub-chains.
    junction_pairs[]:JunctionPair // Array of the pair of the ID's of the junction points
             //                      at the both ends of the sub-chain.
             // This parameter is available only when the type of the polymer is GENERAL.
             // The values of the ID must start from 0.
             // For example, when the number of elements of Block is 1 and a pair of
             // ID's is [0, 0], it means that this polymer is a ring polymer.
}
```

A polymer is defined as a set of sub-chains that are made of monomers and junctions. Although the data structure of the polymer is simple, it can describe any types of chain topologies. The following figure shows an image of the input data for a polymer chain.

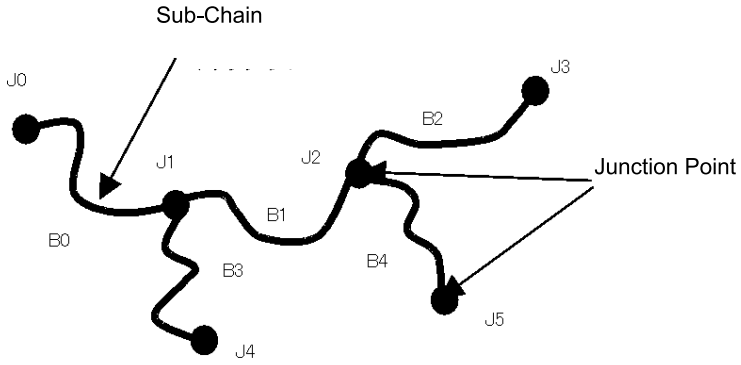


Figure 3.4: An image of the input data for a polymer chain

In this figure, the sub-chains are made of monomers of the same kind. There are junctions at the both ends of each sub-chain. We number all the sub-chains (B0, B1, ···, B4) and all the junctions (J0, J1, ···, J5). The numbers with character B mean sub-chains and those with J mean junctions. Please remember that the numbers should start from 0 and should be sequential. These procedures complete the definition of the topological structures of polymers. The UDF data structure "Block" describes a sub-chain and has "name" and "length". A complete array of the pairs of the junction ID's at both ends of each sub-chains is required. This array is stored in UDF data "JunctionPair". The order of the numbering on the sub-chains and junctions is arbitrary. If the information on the chain topology is correctly given, SUSHI can automatically search for the path that is used in the calculation of the path integrals. If there is a duplication or a lack in the definition of the ID numbers, SUSHI aborts with an error message.

Finally, the UDF data structure "SUSHIInput.components" is defined as an array of polymers and solvents as follows.

```
// Data definition
class Components:{
    polymers[]:Polymer // Array of polymers.
    solvents[]:Solvent // Array of solvents.
}
SUSHIInput:{
    .....
  // Polymer and solvent ##############################
    monomers[]:Monomer // Array of a monomer.
        // Array of the characteristic parameters of monomers.
    monomer_SCF_character_table[]:MonomerSCFChar
        // A set of components (polymers and solvents) that can be used in the simulations.
    components:Components
    .....
}
```

In the sample file, this data is specified as follows.

```
// Data
  components { // Components
    polymers [
      id0 { // Polymer
        type "HOMO"
        blocks [
          id0 { // Block
            monomer_name "A"
```

```
            number_of_monomer 20
          }
        ]
        junction_pairs [
        ]
      }
    .....
    solvents [
    ]
  }
```

You may wonder why no data is contained in "junction_pairs". This is because the structure of the polymer is already specified by "type". For several simple polymer structures, you can specify only with "type" and "Block". For the details, please refer to the comments in the data definition part of the sample file. In this example, since "type" is specified as "homo polymer", only a single sub-chain composed of 20 A-type monomers is specified in the array of subchains, i.e. "blocks". Except for the homo polymers, two or more sub-chains with different kinds of monomers with different lengths are specified in this array. If the polymer type is specified as "GENERAL", you have to specify the topology of the chain by yourself using "junction_pairs". In the present sample file, there is no solvent.

Volume fraction

After specifying the molecules, you have to specify the volume fractions of each molecule. The data structure is as follows.

```
// Data definition
class VolumeFraction:{  // Volume fraction.
    id:int // The element index of the component. The number starts from 0.
    volume_fraction:double
        // If the ensemble is CANONICAL, this parameter specifies the total volume
        // fraction in the system.
        // If the ensemble is GRANDCANONICAL, this parameter specifies the equilibrium
        //  volume fraction in the bulk phase.
    ensemble:string  // Statistical ensemble of the system.
        // CANONICAL : Canonical ensemble.
        // GRANDCANONICAL : Grand canonical ensemble.
        //     Dynamic calculation is not available for this case.
    bulk_volume_fraction:double  // The volume fraction in the bulk phase.
        //     This parameter is used when the volume fractions in the simulation box and
        //     those in the reservoir should be distinguished.
        //     If you do not use this feature, set -1 to this parameter.
}
class VolumeFractions:{
    polymer_volume_fractions[]:VolumeFraction  // Array of the volume fractions of the
                                               // polymers.
    solvent_volume_fractions[]:VolumeFraction  // Array of the volume fractions of the
                                               // solvents.
}
SUSHIInput:{
    .....
    volume_fractions:VolumeFractions
    .....
}
```

The input data structure of SUSHI is designed in such a way that the users can use SUSHI just like an

experimental apparatus. Users of SUSHI can prepare the input file as if they pick up several reagents from a set of polymers and solvents in a shelf, and mix them in a flask. Various simulations can be performed in this way. Therefore, all the polymers and the solvents defined in "SUSHIInput.components" are not necessarily used in the simulation. Please think that "SUSHIInput.components" is a shelf in which polymers and solvents are stored in separate bottles. The users specify the components used in their simulations by specifying the kinds of the components (polymers or solvents), the reagent ID's, and the volume fractions. "id:int // The element index of the components" corresponds to the ID of the reagent. This ID number is equivalent to the element index of the molecule defined in "SUSHIInput.components". For example, if you need the polymer stored in the 1st element of the array "SUSHIInput.components", please specify 0. The order of the elements in the input data can be in an arbitrary order of the array of C language (the index must be started from 0). A reverse order or a discrete order can be accepted. However, multiple definitions of the same id is not allowed. The element indices of the "SUSHIInput.components" are also used in many other input data structures. Please do not confuse them with the element indices defined in volume_fractions below. In the sample file, UDF data "SUSHIInput.volume_fractions" is defined as follows.

```
// Data
  volume_fractions { // VolumeFractions
    polymer_volume_fractions [
      id0 { // VolumeFraction
        Id 0
        volume_fraction 0.5
        ensemble ""
        bulk_volume_fraction -1
      }
      id1 { // VolumeFraction
        Id 1
        volume_fraction 0.5
        ensemble ""
        bulk_volume_fraction -1
      }
    ]
    solvent_volume_fractions [
    ]
  }
```

The volume fractions of A and B polymers are 0.5, respectively. The statistical ensemble is not specified. In such a case, canonical ensemble is set as a default value. The bulk volume fraction is not specified because the value of "bulk_volume_fraction" is -1. The "bulk_volume_fraction" is available only for the grand canonical ensemble. If the grand canonical ensemble is selected and the "bulk_volume_fraction" is not specified, the value of the "volume_fraction" is used as the value of the "bulk_volume_fraction". For a system containing free polymers and grafted polymers, both volume_fraction and bulk_volume_fraction should be specified for the free polymers while only volume_fraction should be specified for the grafted polymers. In such a case, the values of volume_fraction and bulk_volume_fraction are in general different. Note that the convergence of the SCF iterations is dependent on the choice of these parameters. Also note that the dynamic calculations are not available for systems with grand canonical ensemble.

χ parameter

Next, χ parameter is specified. It is easy to specify this parameter as is shown below.

```
// Data definition
class ChiParameter:{
    name_i:string     // The name of the species of i-th monomer.
    name_j:string     // The name of the species of j-th monomer.
    parameter:double  // The value of chi parameter.
```

```
                    // Chi_ij is automatically set using the symmetric relation
                    // Chi_ij = Chi_ji.
                    // If the value is not defined, it is assumed to be 0.
}
SUSHIInput:{
    .....
    chi_parameters[]:ChiParameter  // Array of Chi parameter values.
    .....
}
// Data
  chi_parameters [
    id0 { // ChiParameter
      name_i "A"
      name_j "B"
      parameter 0.2
    }
  ]
```

In the sample file, as is shown above, χ_{AB} are set as 0.2. Since the chain length was previously specified as 20, the phase separation in this system is characterized by $\chi N = 4$.

External conditions

So far, almost all the input items have been specified to set up the system. Finally, we introduce a trick to achieve the fast convergence of the SCF iterations for the equilibrium phase separation structures. The input UDF data structure "SUSHIInput.external_conditions.static_conditions.domain_specification_conditions[]" is prepared for this purpose. This is used to set initial values of the self consistent fields in order to generate domains in desired regions. The data structure is as follows.

```
// Data definition
class AxisRegion:{ // Specify an interval on an axis.
    axis_name:string // Name of the axis: X, Y, Z, R or H.
    r_min:double     // Minimum value of the interval.
    r_max:double     // Maximum value of the interval.
                     // When Maximum = Minimum, it corresponds to a mask on a point.
}
class DomainSpecificationCondition:{
    name:string                 // The name of the segment species.
    domain_regions[]:AxisRegion // The array of the intervals on the axes.
}
class StaticConditions:{
    ....
    domain_specification_conditions[]:DomainSpecificationCondition
        // An array of the conditions for setting the initial values of the
        // self-consistent fields.
    ....
}
class ExternalConditions:{
    ....
    static_conditions:StaticConditions
    ....
}
SUSHIInput:{
    .....
     external_conditions:ExternalConditions    .....
    .....
```

```
}
```

An example of the Input data is as follows.

```
// Data
      domain_specification_conditions [
        id0 { // DomainSpecificationCondition
          name "A"
          domain_regions [
            id0 { // AxisRegion
              axis_name "X"
              r_min 0
              r_max 16
            }
          ]
        }
      ]
```

These input data are meant to produce an "A" domain in the region $x \in [0, 16]$.

We finished the explanation of the input data file, and explain how to use it on GOURMET.

Execution of the calculation

Let us start SUSHI on GOURMET or on a console.

The name of the output file is now assumed to be "interface_uot.udf". Enter the following command to start SUSHI.

```
> sushi -Iinterface_uin.udf
```

After the simulation is finished, open the output file "interface_uot.udf" using GOURMET. The initial window is for the input data. Let us display the simulation results. As shown in the following figure, please choose "Record" and then choose "Step1".

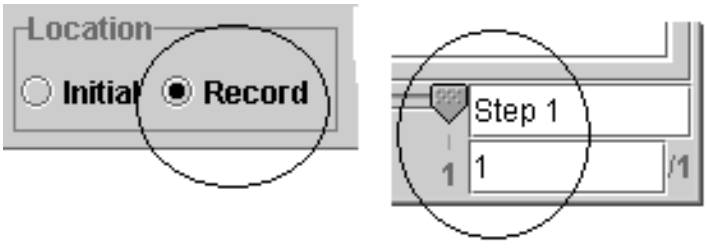


Figure 3.5: Move to the result

Move your mouse cursor to the SUSHIOutput subholder of the left side table and push the right button of the mouse. You can choose plot_1D_field button to plot the result using Gnuplot as shown in the next figure.

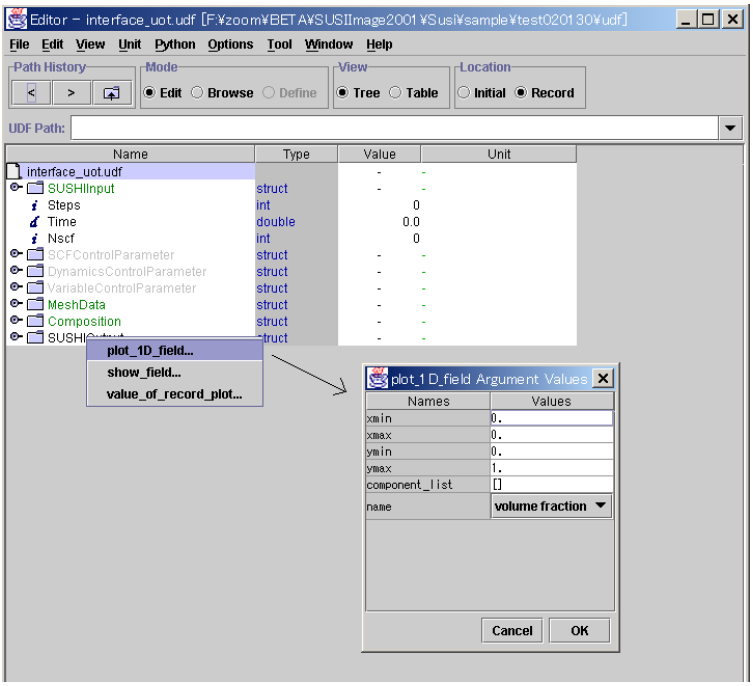


Figure 3.6: Display the simulation result

Using the sample UDF file, an interface of an A/B polymer blend system is simulated. The SUSHI is suitable for calculating such interfaces. By modifying the input UDF data, you can perform various simulations. In the following, some examples will be given.

3.2.2 Phase separation

Let us modify the boundary condition "SUSHIInput.boundary_condition" in the input UDF file "interface_uin.udf".

```
// Data
  boundary_condition { // BoundaryCondition
    conditions [
      id0 { // AxisBoundaryCondition
        axis_conditions [ "PERIODIC"  ]
      }
    ]
  }
```

We have changed the boundary conditions from reflective boundary conditions to periodic boundary conditions. This modified input UDF file is stored in "blend_4D_dy_uin.udf". The simulation result produced with this input UDF file is a phase-separated state where two interfaces are formed due to the periodic boundary conditions. The result is shown in the next figure.

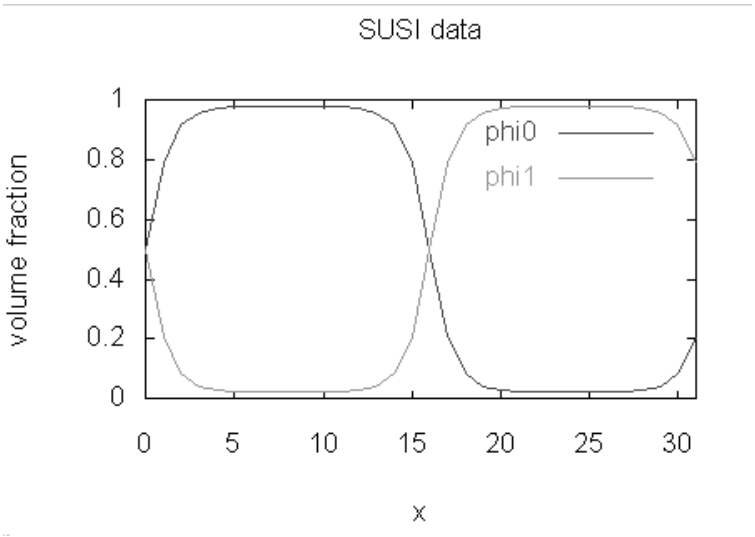


Figure 3.7: An interface of an A/B polymer blend with periodic boundary conditions, $\chi N = 4$.

3.2.3 Lamellar structure

Next, let us calculate a lamellar structure of a block copolymer melt. The data of the variable "SUSHI-Input.components.polymers" are modified to produce an input UDF file "blend_uin.udf", where an A homopolymer and a B homopolymer are connected to form a diblock copolymer.

Since the system is composed of a single type of molecule, the value of the volume fraction in "SUSHI-Input.volume_fractions.polymer_volume_fractions[]" should be changed to unity for the species id0. As the order-disorder point of the block copolymer melt is different from that of the polymer blend, you have to increase the value of χN, for example $\chi = 0.4$ ($\chi N = 16$). The domain specification specified in "SUSHIInput.external_conditions.static_conditions.domain_specification_conditions[]" should be canceled.

```
// Data
    .....
    polymers [
      id0 { // Polymer
        type "BLOCK"
        blocks [
          id0 { // Block
            monomer_name "A"
            number_of_monomer 20
          }
          id1 { // Block
            monomer_name "B"
            number_of_monomer 20
          }
        ]
        junction_pairs [
        ]
      }
```

The modified input UDF file is "block_uin.udf". The simulation result for this input UDF file is as follows. Obviously, a lamellar structure is obtained. As the system size is not optimized to the lamellar spacing, the obtained lamellar structure may not be the globally stable equilibrium state with the minimum free energy.

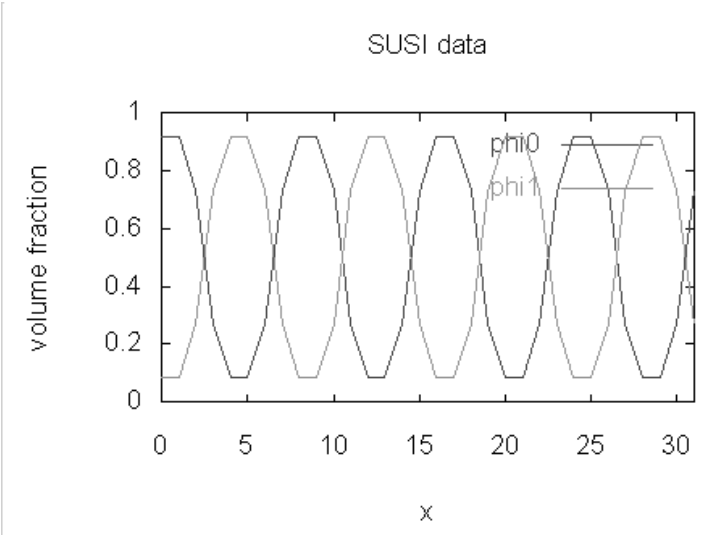


Figure 3.8: Lamellar structure of a block copolymer melt with $\chi N = 16$.

3.2.4 Effect of solid wall

Next, let us simulate a polymer solution near a solid wall. This can be performed by slightly modifying the contents of "interface_uin.udf". First, the boundary conditions (NEUMANN, NEUMANN) should be changed to (WALL or DIRICHLET, NEUMANN), the former corresponding to the solid wall. To simulate the polymer solution, set the chain lengths of A and B as 100 and 1, where the polymer B corresponds to the solvent. Instead of specifying the solvent as a polymer with length 1, you can also specify it as a solvent directly. Now we choose the value of χ_{AB} as 0.5 so that the solution corresponds to a θ-solvent. In order to simulate a semi-dilute solution, the volume fractions of A and B are set to 0.001 and 0.999, respectively. The most important modification of the input UDF data is the change of the statistical ensemble and the volume fraction. We should use the grand canonical ensemble instead of the canonical ensemble, and have to specify the "bulk_volume_fraction" if necessary. The input UDF data thus obtained are as follows.

```
// Data
  volume_fractions { // VolumeFractions
    polymer_volume_fractions [
      id0 { // Volume fraction
        Id 0
        volume_fraction 0.001
        ensemble "GRANDCANONICAL"
        bulk_volume_fraction -1
      }
      id1 { // VolumeFraction
        Id 1
        volume_fraction 0.999
        ensemble "GRANDCANONICAL"
        bulk_volume_fraction -1
      }
    .....
  }
```

The statistical ensemble is specified as GRANDCANONICAL. As we do not specify the "bulk_volume_fraction", the values of the "volume_fraction" at the reflective boundary are automatically used as the values of the "bulk_volume_fraction". Then the simulation result will be an grad canonical equilibrium state in contact with the bulk state. The modified input UDF data are stored in the file "depletion_uin.udf". The simulation result is as follows.

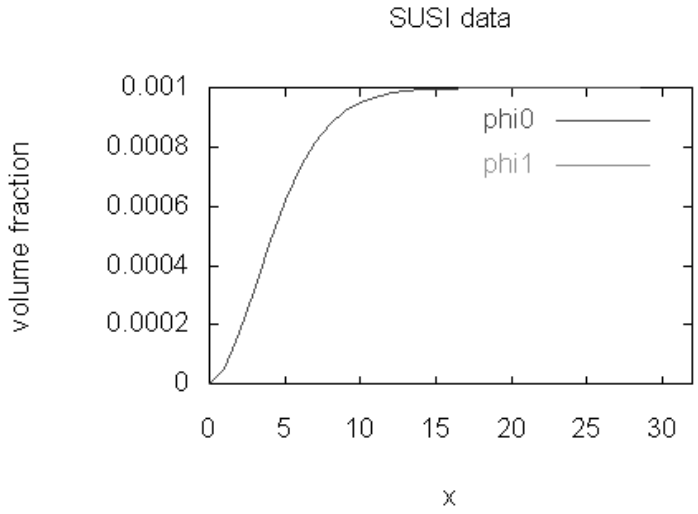


Figure 3.9: Depletion near a solid wall

The chains are excluded from the region close to the solid wall, and a depletion zone is formed. This is due to the entropic effect of the polymer conformation that does not like to sit in the vicinity of the solid wall.

3.2.5 Adsorption

If there is an attractive interaction between a solid wall and polymers, what will happen? Let us perform a simulation of such a system. The input UDF data can easily be modified so that the interaction between the wall and the polymer is introduced.
The parameter "SUSHIInput.external_conditions.surface_chi_parameters[]" is specified as follows.

```
// Data definition
class SurfaceChiParameter:{
    boundary_name:string // The name of the boundary wall that produces the interaction.
    target_name:string   // The name of the target segment species.
    parameter:double     // The value of the chi parameter.
}
class ExternalConditions:{
    surface_chi_parameters[]:SurfaceChiParameter
    ....
}
SUSHIInput:{
    .....
    external_conditions:ExternalConditions    .....
    .....
}
// Data
  external_conditions { // ExternalConditions
    surface_chi_parameters [
      id0 { // SurfaceChiParameter
        boundary_name "XMin"
        target_name "A"
        parameter -2
      }
    ]
    ....
```

```
}
```

The interaction parameter between YZ surface at X= 0 and the A-type segment is set as −2. The modified input UDF data are stored in "adsorption_uin.udf". The simulation result is shown in the following figure. While the polymers formed a depletion layer in the previous example, such a depletion layer disappears in the present example due to the adsorption of the polymers to the solid wall.

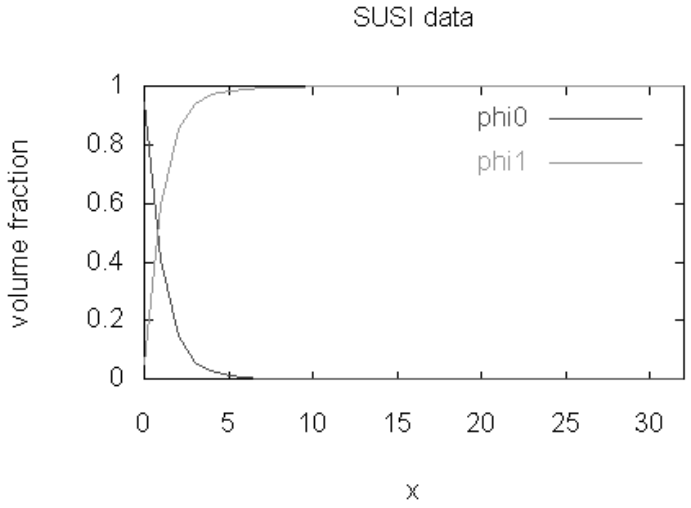


Figure 3.10: Adsorption of polymers to a solid wall

3.2.6 Grafted chains

If the chains are grafted rather than are adsorbed to a solid wall, what will happen? Let us modify the input UDF file "depletion_uin.udf" so that the one end of the A polymer is grafted to the wall. Additional UDF data for the grafting condition are "SUSHIInput.external_conditions.graft_conditions[]".

```
// Data definition
class GraftCondition:{
    polymer_ID:int       // ID of the grafted polymer.
    junction_ID:int      // ID of the grafted free end.
    boundary_name:string // The name of the boundary onto which the polymer is grafted.
    obstacle_ID:int
}
class ExternalConditions:{
    ....
    graft_conditions[]:GraftCondition
    ....
}
SUSHIInput:{
    .....
     external_conditions:ExternalConditions    .....
    .....
}
// Data
    graft_conditions [
      id0 { // GraftCondition
        polymer_ID 0
        junction_ID 0
        boundary_name "XMin"
```

```
    }
  ]
```

In this case, the free end with ID 0 of the polymer with ID 0 is grafted to the YZ surface at $X = 0$. Simultaneously, the UDF data "SUSHIInput.volume_fractions.polymer_volume_fractions[]" should be modified as follows.

```
// Data
  volume_fractions { // VolumeFractions
    polymer_volume_fractions [
      id0 { // VolumeFraction
        Id 0
        volume_fraction 0.001
        ensemble "CANONICAL"
        bulk_volume_fraction -1
      }
      id1 { // VolumeFraction
        Id 1
        volume_fraction 1.
        ensemble "GRANDCANONICAL"
        bulk_volume_fraction -1
      }
    .....
  }
```

The statistical ensemble for the polymer A should be CANONICAL, because the grafting polymer cannot escape from the system and therefore the exchange between the system and the bulk phase is inhibited. The volume fraction of the polymer B is changed to 1.0 from 0.999. This means that the bulk phase is composed of only B polymers, and all the A polymers are grafted to the wall. The modified data are stored in "graft_uin.udf". When you perform the simulation run, the SUSHI will give you a warning telling you that the sum of the volume fractions is not equal to 1.0. You can neglect this warning. The simulation result is shown in the next figure.

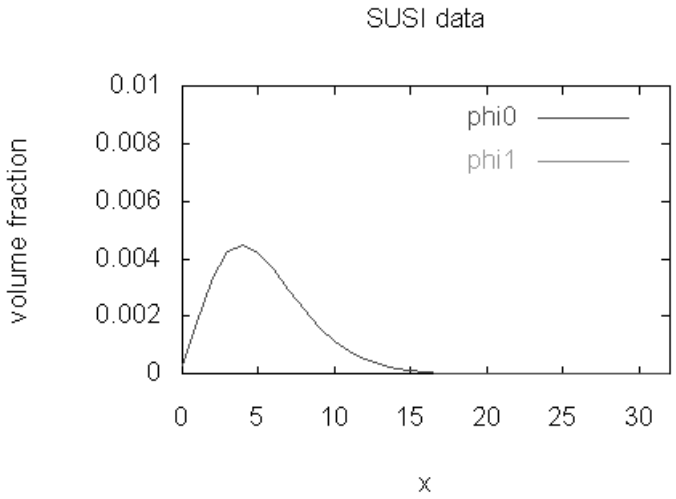


Figure 3.11: Graft

Due to the grafting condition, the segments of the polymer are distributed near the wall with a depletion layer in the vicinity of the wall surface.

3.3 Example of two dimensional simulations

Next we explain how to perform two-dimensional simulations.

3.3.1 Phase separation dynamics of an A/B polymer blend system

Let us retry the simulation of the A/B polymer blend with $\chi N = 4$ in a system with the periodic boundary conditions, which was discussed at the beginning of Section 3.2.1. An extension to a two-dimensional system is easy. To realize this, you have only to add an extra "Axis" to "SUSHIInput.mesh.axes[]". The input UDF data stored in "blend2D_4_dy_uin.udf" is for a simulation of an A/B polymer blend system with 32×34 meshes. (The shape of the system is intentionally chosen as non-square shape so that we can check whether the two-dimensional system is appropriately treated.) In order to accelerate the simulation runs, we choose a shorter polymer length of 10 and larger χ parameter value of 0.4. However, the system is essentially the same as that described by "blend_uin.udf". Only exception is that the type of "SUSHIInput.calculation_method" is set to DYNAMICS in order to perform a dynamic simulation. In this case, we can choose a rather large value 1.0 for the parameter "constW". This is because the segment densities ϕ's are fixed during the SCF iterations at every time step. When you specify the dynamic simulation, the UDF parameter "SUSHIInput.calculation_method.DYNAMICS" becomes effective. The actual format and the data for this parameter are as follows.

```
// Data definition
class DynamicsParameter:{
    delta_t:double              // The time mesh width.
    variable_delta_t:VariableDelt // Parameters for variable time mesh.
    max_dynamics_step:int       // The total numbers of the time steps.
    output_interval_step:int    // Output step interval.
    archives_interval_step:int  // Output step interval to the archives file.
    log_interval_step:int       // Output step interval to the standard output file.
    dynamics_scheme:select { "EXPLICIT", "EXPLICIT2", "IMPLICIT" } // EXPLICIT is the default value.
        // The scheme for the integration of the equation of motion.
        // EXPLICIT  The explicit scheme. Usually, this scheme is enough. Euler scheme.
        // EXPLICIT2 The 2nd explicit scheme. 2step Runge-Kutta scheme.
        // IMPLICIT  The implicit scheme.
    compressibility:double      // Compressibility for SCF
}
class CalculationMethod:{
    type:select { "STATICS", "DYNAMICS", "MONTECARLO" }
    DYNAMICS:DynamicsParameter  // Refer to Section 2.7.10.
    MONTECARLO:MonteCarloParameter
}
SUSHIInput:{
    calculation_method:CalculationMethod
    .....
}
// Data
SUSHIInput:{ // SUSHIInput
  ....
  calculation_method { //CalculationMethod
    type "DYNAMICS"
    DYNAMICS { //DynamicsParameter
      delta_t 0.01
      max_dynamics_step 500000
      output_interval_step 5000
      archives_interval_step 5000
      log_interval_step 1
      dynamics_scheme ""
```

```
      compressibility 0
    }
    .....
  }
  ....
}
```

The important parameters in the dynamic simulations are the time mesh width, the total numbers of the time steps, and the output step interval. In the above example, we did not specify the "dynamics_scheme". In such a case, the explicit scheme is used as a default, which is sufficient for almost all the cases. Time mesh width is the most important parameter. If this parameter is chosen to be too large, the simulation will soon diverge. If it is too small, the time evolution becomes accordingly slow. Although the value of the time mesh width is in general depends on the spatial mesh width, the time mesh width of 0.001 is usually a good selection for the spatial mesh width 1. In dynamic simulations, one can specify the mobilities of the polymer segments and the solvents by "polymer_mobilities" and "solvent_mobilities" in the UDF data "SUSHIInput.external_conditions.dynamic_conditions". When a very low-concentration component exists in the system, these mobilities should appropriately be specified. For the details, readers should refer Sections 2.6 and 2.7.10. In the following example, we do not specify these parameters. If the value of the compressibility is larger than 0, the compressible dynamics is enable.

```
// Data definition
class SegmentMobility:{  // Mobility of a segment.
    segment_name:string  // The name of the segment/solvent whose mobility is specified.
    mobility:double      // The value of the mobility.
}
class LocalMobility:{  // The position dependent mobility.
    component_ID:int   // ID of the component.
    type:select { "ROUSE", "REPTATION" }
}
class DynamicConditions:{
    segment_mobilities[]:SegmentMobility
    types_of_polymer_mobility[]:LocalMobility
    types_of_solvent_mobility[]:LocalMobility
    ....
}
class ExternalConditions:{
    ....
    dynamic_conditions:DynamicConditions
}
SUSHIInput:{
    .....
    external_conditions:ExternalConditions    .....
    .....
}
```

Now, open the simulation result stored in "blend_dy_4_uot.udf" using GOURMET. Move your mouse cursor to the SUSHIOutput subholder of the left side table and push the right button of the mouse. You can choose "show_field..." button to plot the result using GOURMET viewer. Clicking the "animation button" in the lower left, you can obtain an animation of the time evolution. As is shown in the following figure, the time evolution of the phase separation of an A/B polymer blend will be shown.

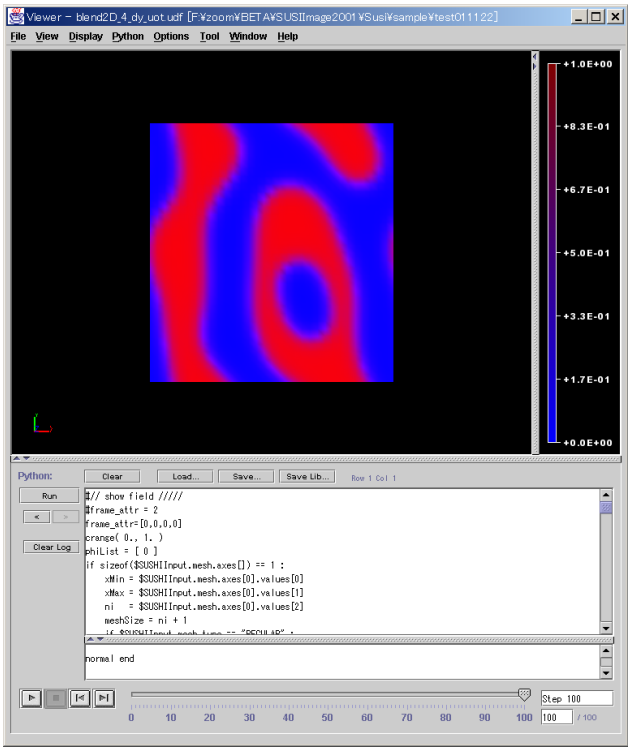


Figure 3.12: Dynamics of an A/B polymer blend

3.4 Example of three dimensional simulations

Finally, we show an example of three dimensional simulations. As you may guess, the extension to a three dimensional system can be done by adding one more mesh axis.

3.4.1 Cylindrical structure

As an example of the three dimensional simulations, let us simulate a domain morphology of a micro phase separation. It is known that a block copolymer with the block ratio 0.25 and with $\chi N = 20$ shows cylindrical structure. The mean field simulation by using SCF can reproduce such a cylindrical morphology. However, the system can easily be trapped in a metastable state that corresponds to a local minimum of the free energy functional. Thus, a perfect domain morphology without defects is not obtained by simply performing a simulation from a uniform state as an initial condition. The most efficient method to realize the target perfect morphology is to use the UDF data used in the example of the beginning of the section for the static calculation, i.e. "SUSHIInput.external_conditions.static_conditions.domain_specification_conditions[]". Using this method, we can simulate the cylindrical structure as you can confirm using "cylinder_3D_uot.udf". If you look at the input file "cylinder_3D_uin.udf", you can easily understand what we have specified in this input UDF file. As is shown in the following figure, the simulation result can be visualized with use of "show.py" on the "Viewer" of GOURMET. We can clearly observe a set of well aligned cylindrical domains. Since the optimization of the periodicity of the cylindrical domains is not performed, the hexagonal cylinder structure shown in the figure may not be the equilibrium state that has the minimum free energy. However, we expect that you can understand how to create a desired domain morphology. If you use the "restart" function of SUSHI, the optimization of the domain periodicity can also be performed.

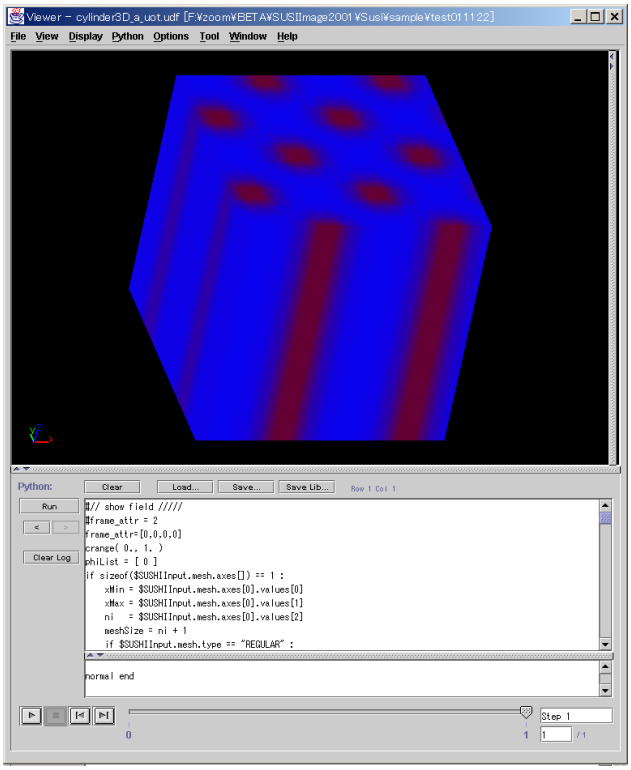


Figure 3.13: Cylindrical structure of a block copolymer melt

3.5 Final remarks

Although all the simulations shown in this chapter are performed on regular meshes, SUSHI can also perform simulations on one dimensional spherical coordinates, two dimensional cylindrical coordinates, and so on. The input data are not so different from those for the regular mesh. Moreover, many other functions are also implemented in SUSHI. You can simulate an infinite variety of systems by combining the functions of SUSHI. If you have any questions, you are encouraged to send a query to `octa.jp BBS` for help. We hope that SUSHI can be a useful tool for your research.

Chapter 4

Sample problems

4.1 Calculation of interfacial tension for an A/B polymer blend

4.1.1 Outline

In this section, we show results of a study on the polydispersity effects on the interfacial tension of an A/B homo polymer blend. Interfacial tension is defined as the excess free energy accumulated at the interface. It is generally calculated as the difference between the total free energy of the target system with an interface and the sum of the free energy values of individual uniform equilibrium systems that composes the two bulk phases of the target system. By using SUSHI, the excess free energy can automatically be obtained from a grand canonical SCF calculation if the segment density of each component in the equilibrium bulk phase is known. Thus, the segment density in the bulk equilibrium phase must be obtained beforehand using coarse-grained theories such as the Flory - Huggins mean field theory. This constraint limits the applicability of the grand canonical SCF method to the multi-component blend systems. In this study, instead of the grand canonical SCF calculation, we use the canonical SCF calculation to evaluate the interfacial tension, and we will confirm the applicability of the method.

4.1.2 System and parameters

The effects of polydispersity on the polymer interfaces are investigated for A/B binary homopolymer mixtures where both polymers have molecular weight distributions. The equilibrium structure of such a polymer blend is obtained by the 1-dimensional static SCF calculation under the Neumann boundary condition. Now, let the index $K = A, B$ specify the segment species. We assume that each polymer component ($K = A, B$) is subdivided into several kinds of chains specified by another index i each of which has different chain length. Then, in this calculation, the parameters that determine the state of the system are the segment interaction parameter χ_{AB}, the volume fraction ϕ^0_{Ki} and the total chain length N_{Ki} of the i-type chain of the K-type polymer. In addition, the equilibrium structure can efficiently be calculated by setting up the initial value of the self-consistent field using external conditions. Readers should refer to Section 2.9.13 of the SUSHI manual. Since the segment density profile of each component and the free energy of the equilibrium state are output at the end of the calculation by SUSHI, the excess free energy of the system can be evaluated with the following procedures.

4.1.3 Calculation procedure

Let us denote the segment density of the Ki type chains in the α-type bulk phase as ϕ^α_{Ki}, where α specifies each of the coexisting equilibrium phases. Using the output data of these ϕ^α_{Ki} and the equilibrium free energy $\mathcal{F}$, one can calculate the free energy of the bulk phase f^{bulk} and the equilibrium chemical potential of each component μ_{Ki}. Then, the excess free energy $\mathcal{F}_{excess}$ is calculated as follows.

$$f^{bulk} = \sum_i \frac{\phi^\alpha_i}{N_i} \ln \frac{\phi^\alpha_i}{N_i} + \frac{1}{2} \sum_{ij} \chi_{ij} \phi^\alpha_i \phi^\alpha_j \tag{4.1}$$

$$\mu_i = 1 + \ln \frac{\phi^\alpha_i}{N_i} + N_i \sum_j \left(\chi_{ij} \phi^\alpha_j - \frac{\phi^\alpha_j}{N_j} \right) - \frac{1}{2} N_i \sum_{jk} \chi_{jk} \phi^\alpha_j \phi^\alpha_k \tag{4.2}$$

$$\mathcal{F}_{excess} = L\mathcal{F} - Lf^{bulk} - L\sum_i \frac{\mu_i\left(\phi_i^0 - \phi_i^\alpha\right)}{N_i}. \tag{4.3}$$

Here, we redefined the indices i, j, and k in these equations so that they specify all types of the polymer chains in the blend system (i.e. i component of K polymer, eg. A1, A2, $\cdots$, B1, B-2, and $\cdots$), and the parameter L expresses the system size in the SCF calculation. The above calculation procedure can automatically be performed by a Python script. On the other hand, using the interfacial problems as a target, we show three examples of the extensions of SUSHI described in the appendices. Source programs for these extended simulators are stored under the "InterfaceSimulator" directory. The names of these simulators are

- FluidSimulator(fluid)
- MicelleSimulator(micelle)
- SurfaceSimulator(surface).

A set of calculation procedures for the interfacial tension is packaged in the input UDF file named "fluid.udf" for the FluidSimulator. Please refer to the appendix D

4.1.4 Effect of polydispersity

Broseta[22], Anastasiadis[23] and their coworkers evaluated the free energy of an immisible homo polymer blend based on the Flory-Huggins model combined with random phase approximation (RPA), and by experiments. They expressed the polydispersity effects on the interfacial tension in terms of the number-averaged molecular weight of each polymer component. Using SUSHI, we calculated the effects of polydispersity on the polymer interfaces of a simple A/B binary homopolymer mixture where both polymers have the same bimodal molecular weight distribution. For such a system, the value of the excess free energy F_{excess} is obtained by the above method, from which we investigated its dependence on the number average molecular weight by changing the chain length and the volume fraction. We found that each system shows almost the same behavior, which resembles to that of the molecular weight dependence of a mono disperse polymer blend system. Moreover, as is expected, we confirmed that it asymptotically approaches the theoretical value of the interfacial tension obtained by Helfand and Tagami[24] in the limit of infinite molecular weight. These calculation results are not in contradiction to the theoretical prediction and the experiment. Therefore, our proposed calculation scheme is considered to be a more flexible method than the one using the grand canonical ensemble. On the other hand, the excess density profiles of both the short chain and the long chain components show that the short chains are accumulated near the interface and depletion of a long chain takes place. Therefore, the SCF analysis on the structure of the interfaces in polydisperse polymer blends is quite useful in improving the physical properties of the interfaces (interfacial tension, thickness, adhesion, etc.). Although it is a target of the future research, an extension of the current static calculation to dynamic processes using the dynamic mean field calculation will also be efficient and useful.

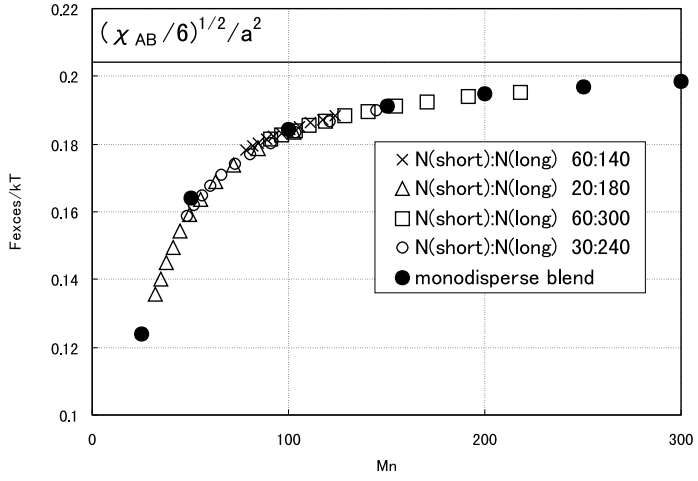


Figure 4.1: Molecular weight dependence of the excess free energy ($\chi_{AB} = 0.25$)

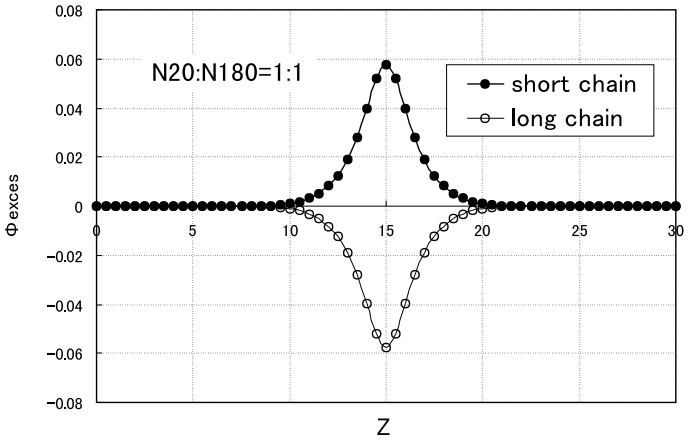


Figure 4.2: Excess segment density profiles of short and long chains ($\chi_{AB} = 0.25$)

4.2 Calculation of micellar distribution of an A-B diblock copolymer

4.2.1 Outline

When an A-B block copolymer is dissolved into a selective solvent, molecular aggregates called micelles are formed. Usually, polydispersed micelles are formed when the concentration of the polymer exceeds a certain critical micelle concentration(cmc).

Here, we will describe how to calculate the size distribution of the micelles in a diblock copolymer solution at a given concentration. In order to simplify the problem, we assume that the system is composed of an AB diblock copolymer and a solvent C, and also assume that the micelles are spherical. We can simulate a certain amount of the AB block copolymer dissolved into the solvent C by using the canonical ensemble where the volume fraction of each component is kept constant. When the concentration of the block copolymer is considerably high, such a calculation method can reproduce a regular structure called a liotropic liquid crystal. However, the method is not applicable to a micellar solution whose concentration is close to cmc. This is because the micelles can not be spontaneously formed due to a large free energy barrier in forming a micelle with the critical size. Therefore, in order to simulate a micellar solution, the following two procedures are needed.

1. First, we have to calculate the structure of an isolated micelle with all possible sizes by SUSHI and determine the value of the excess free energy for each of these micelles.

2. The free energy of a micellar solution is defined as a function of the distribution of the micellar size, and the free energy is minimized with respect to such a distribution of the micellar size under the condition that the concentration of the whole polymers is fixed.

Details of these procedures are given below.

4.2.2 Calculation of an isolated micelle

When treating an isolated micelle, we need to keep the aggregation number of the micelle constant. For such a purpose, we calculate the block copolymer using the canonical ensemble and the solvent using the grand canonical ensemble, respectively, with the bulk concentration of the solvent to be 1.0. Furthermore, in order to obtain a stable micelle, the ends of the subchains that dislike the solvent are restricted in a certain region at the center of the micelle. Such a calculation can be done using the MASK function currently implemented in SUSHI. If the region to which the chain ends are restricted is too small, the free energy of the system will be excessively large because the numbers of the possible conformations of the chains are strongly limited. In order to estimate the suitable size of the micelle, several trial and error simulations are needed. The example of the static SCF calculation for an isolated micelle with the aggregation number $p = 30$ is shown for a

system composed of a block copolymer $A_{40}B_{10}$ and a solvent C. The χ parameters are set as $\chi_{AB} = 1.0$, $\chi_{BC} = 0$, and $\chi_{AC} = 1.2$, respectively. The structure of the calculated micelle is shown in fig. 4.3.

Let us denote the excess free energy of a micelle divided by its aggregation number p as $f(p)$. This $f(p)$ is calculated using the above-mentioned procedure for each aggregation number p, and the result is shown in fig. 4.4. We observe that $f(p)$ takes its minimum value at $p = 29$. This value $p = 29$ gives an estimate of the optimal micellar size. Strictly speaking, one has to minimize the free energy of the whole micellar solution taking the translational entropy of micelles into account. The following paragraph describes such a procedure.

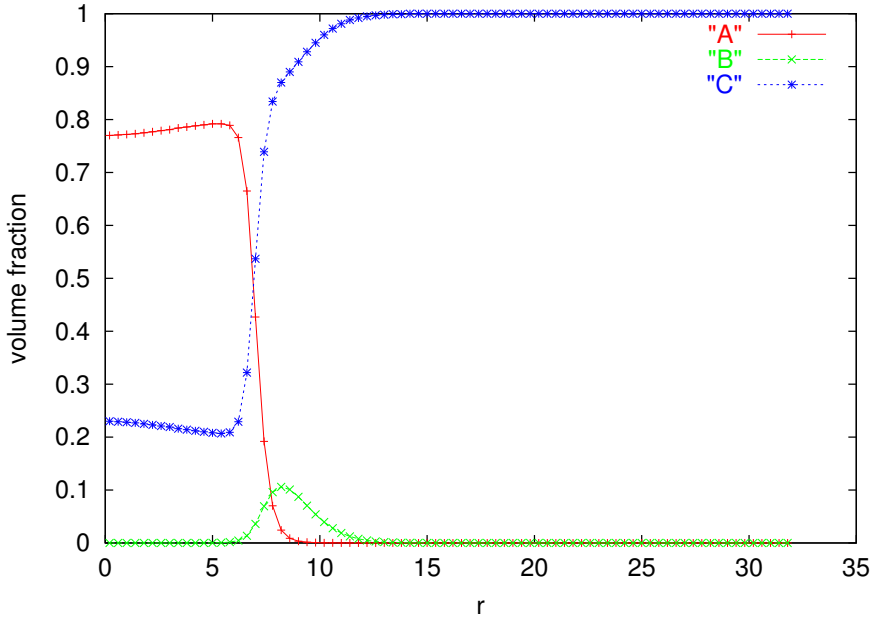


Figure 4.3: Segment density profiles of a micelle with the aggregation number 30.

4.2.3 Calculation of the micellar size distribution

The free energy of a micellar solution is given by the following equation.

$$F\left(\{\phi_p\}\right)/k_BT = \sum_p \phi_p \left(\frac{1}{Np}\ln\phi_p + f(p)\right). \tag{4.4}$$

Here, ϕ_p expresses the distribution of the volume fraction of the micelles with the aggregation number p, i.e. the micellar size distribution in the micellar solution. To obtain the optimum distribution, we have to minimize Eq. (4.4) under the constraint that the total polymer volume fraction $\phi = \sum_p \phi_p$ is fixed. For example, using $f(p)$ shown in fig. 4.4, we can obtain the micellar size distribution as shown in fig. 4.5 for $\phi = 0.005$ and 0.04. Even if the volume fraction of the total copolymer is increased by a factor of 10 or more, the volume fraction of the isolated copolymer chains is almost unchanged. Such a concentration of the isolated copolymers is equivalent to the critical micelle concentration. Therefore, we conclude that the characteristic behavior of micellar formation is well reproduced by this method.

Using a Python script, we can perform the calculation of the distribution $\{\phi_p\}$. In the script, a Lagrange multiplier μ is introduced and the distribution is optimized by minimizing $F\left(\{\phi_p\}\right) - \mu(\sum_p \phi_p - \phi)$ for given μ. In principle, μ is a unique function of ϕ. In the present Python script, however, we simply calculate ϕ as well as the micellar size distribution for a given value of μ. Therefore, in order to obtain the micellar size distribution at a given volume fraction ϕ, one needs to adjust manually the value of μ running the script repeatedly.

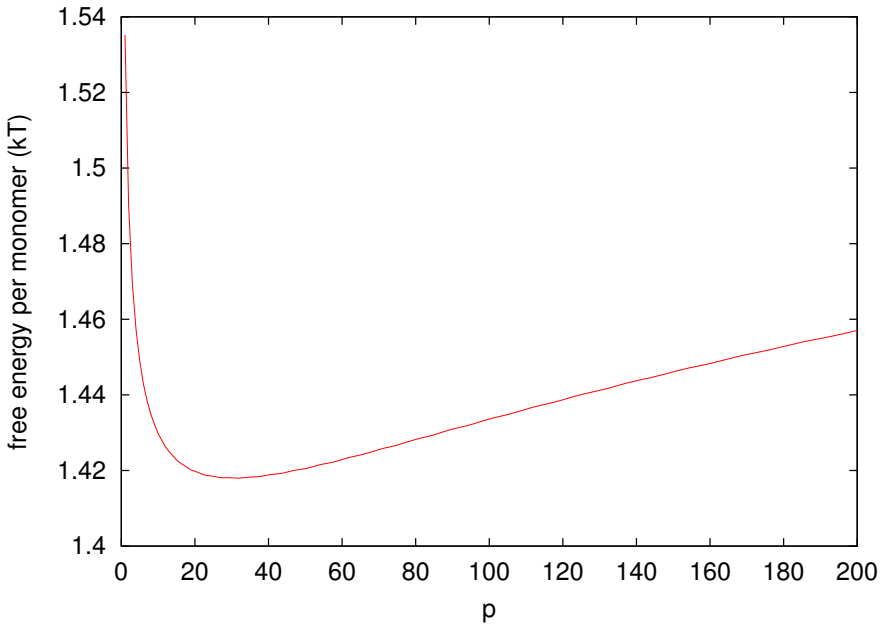


Figure 4.4: Free energy per a copolymer chain in a micelle with the aggregation number p.

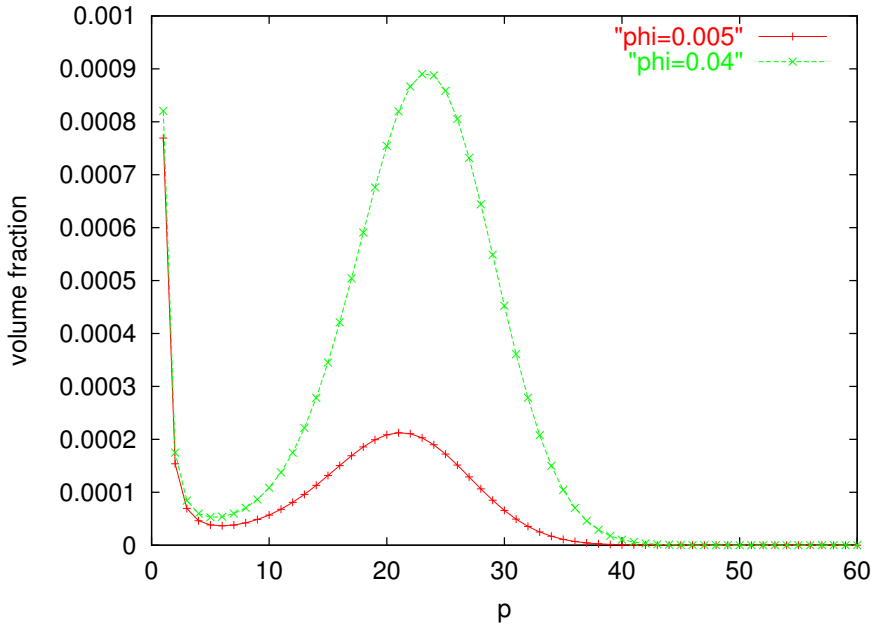


Figure 4.5: Distribution of the aggregation number of micelles in a micellar solution

4.3 Sample UDF list

Here, the contents of the sample UDF files are explained.

ab_ring_uin.udf / uot.udf
1D static calculation of a ring A/B block copolymer.

adsorption_uin.udf / uot.udf
Adsorption of a polymer to a solid wall explained in Chapter 3.

blend_uin.udf / uot.udf
Interface of an A/B polymer blend explained in Chapter 3.

blend2D_4_ADF_dy_uin.udf / uot.udf
Simulation result of a 2D dynamic mean-field calculation of the Approximate Density Functional type model on the phase separation of an A/B polymer blend. This method is under development.

blend2D_4_FH_dy_uin.udf / uot.udf
Simulation result of a 2D dynamic mean-field calculation of the Flory-Huggins type model on the phase separation of an A/B polymer blend explained in Appendix C.

blend2D_4_HYBRID_uin.udf / uot.udf
Simulation result of a 2D calculation of the phase separation of an A/B polymer blend by using the hybrid theory.

blend2D_4_RPA_uin.udf / uot.udf
Simulation result of a 2D calculation of the phase separation of an A/B polymer blend by using the GRPA.

blend2D_4_RPA_HYDRO_uin.udf / uot.udf
Simulation result of a 2D calculation of the phase separation of an A/B polymer blend by using the GRPA and hydrodynamics.

blend2D_4_dy1_uin.udf / uot.udf
Simulation result of a 2D calculation of the phase separation of an A/B polymer blend explained in Chapter 3.

blend2D_4_dy1_comp_uin.udf / uot.udf
Simulation result of a 2D calculation of the phase separation of an A/B polymer blend with compressibility.

blend2D_4_dyr1_uin.udf / uot.udf
Simulation result of a 2D calculation of the phase separation of an A/B polymer blend by using the ϕ-dependent mobilities.

blend2D_4_dyr1_const_uin.udf / uot.udf
Restart of blend2D_4_dyr1_uot.udf

blend2D_4_dyr1_rest_uin.udf / uot.udf
Continue of blend2D_4_dyr1_uot.udf

blend2D_4_dy1r_shear_uin.udf / uot.udf
Simulation result of a 2D calculation of the phase separation of an A/B polymer blend with shear.

block_uin.udf / uot.udf
1D static calculation of the lamella structure of a block copolymer melt explained in Chapter 3.

block2D_7_dy1_uin.udf / uot.udf
2D dynamic calculation of the phase separation of a block-copolymer melt.

block2D_7_lamella_uin.udf / uot.udf
2D static calculation of the lamella structure of a block copolymer melt.

block2D_8_dy1_lo_uin.udf / uot.udf
2D dynamic calculation of the phase separation of a block-copolymer melt with system size optimization.

block2D_8_dy1_lo_comp_uin.udf / uot.udf

2D dynamic calculation of the phase separation of a block-copolymer melt with system size optimization and compressibility of system.

block2D_8_dy1_lo_comp2_uin.udf / uot.udf

2D dynamic calculation of the phase separation of a block-copolymer melt with system size optimization and compressibility in SCF.

block2D_8_dy1_lo_comp_comp_uin.udf / uot.udf

2D dynamic calculation of the phase separation of a block-copolymer melt with system size optimization, compressibility of system, and compressibility in SCF.

block2D_8_lo_uin.udf / uot.udf

2D static calculation of the phase separation of a block-copolymer melt with system size optimization.

block_el_uin.udf / uot.udf

1D static calculation of the lamella structure of a block copolymer melt, ends of which are strong polyelectrolyte.

block_el3_uin.udf / uot.udf

1D static calculation of the lamella structure of a block copolymer melt, ends of which are strong polyelectrolyte with dielectric constants of segment.

comb2D_16_uin.udf / uot.udf

2D static calculation of the equilibrium structure of a comb copolymer melt.

comb3D_16_uin.udf / uot.udf

3D static calculation of the equilibrium structure of a comb copolymer melt.

cylinder3D_a_uin.udf / uot.udf

3D static calculation of the cylinder structure of a block copolymer melt by the domain specification method. Explained in Chapter 3.

depletion_uin.udf / uot.udf

Depletion of a polymer near a solid wall explained in Chapter 3.

graft_uin.udf / uot.udf

A polymer grafting to a wall explained in Chapter 3.

graft_dy_step0_uin.udf / uot.udf, graft_dy_uin.udf / uot.udf

A sample input UDF for the dynamics of a graft reaction. Restart calculation is performed according to the input UDF file graft_dy_step0_uot.udf.

interface_uin.udf / uot.udf

Interface of an A/B polymer blend explained in Chapter 3.

micelle_uin.udf / uot.udf

1D static calculation of a micelle.

montecarlo_uin.udf / uot.udf

2D Monte Carlo calculation of a comb copolymer in a solution.

nonpolyd_uin.udf / uot.udf, polyd_uin.udf / uot.udf

1D static calculation for polydisperse polymers near a solid wall. In the "nonpolyd" case, the symmetry in the path integral is not taken into account in the calculation. One can use these two UDF files to compare the efficiency of the calculation using the symmetry in the path integrals.

quench_uin.udf / upm.udf / uot.udf

Simulation result of a 2D calculation of the phase separation of an A/B polymer blend with time dependent χ parameter. The χ parameters are recorded in the parameter file.

solventSol_eq_uin.udf / uot.udf

2D static calculation of the equilibrium state of a mixture of a solvent and a polymer.

star2D_8_uin.udf / uot.udf

2D static calculation of the equilibrium state of a star-type copolymer melt. The Python script show3color.py can display the density distributions of the three components with different colors.

star3D_8_uin.udf / uot.udf

3D static calculation of the equilibrium state of a star-type copolymer melt. The Python script show3color.py can display density distributions of the three components with different colors.

taper2D_12_uin.udf / uot.udf

2D static calculation of the equilibrium state of a tapered polymer melt.

taper3D_12_uin.udf / uot.udf

3D static calculation of the equilibrium state of a tapered polymer melt.

testRg_uin.udf / uot.udf

An example of the calculation of R_g. 1D static calculation of R_g of a chain whose one end is fixed at a point in a solvent.

testRg2_uin.udf / uot.udf

An example of the calculation of R_g. 1D static calculation of R_g of a full chain whose one end is fixed at a point in a solvent.

triblock2D_7_uin.udf / uot.udf

2D static calculation of the equilibrium structure of a triblock copolymer melt. The Python script show3color.py can display the density distributions of the three components with different colors.

EZ1a_uin.udf / uot.udf

2D dynamic calculation under a weak external electric field. Treatment of the weak effect of external electric field.

EZ2a_uin.udf / uot.udf

2D dynamic calculation under an external electric field. General treatment of the effect of external electric field.

EZ3a_uin.udf / uot.udf

2D static calculation under an external electric field. General treatment of the effect of external electric field.

particle1_uin.udf / uot.udf

2D static calculation with a particle.

particle2_uin.udf / uot.udf

2D static calculation with particles.

particle2B_uin.udf / uot.udf

3D static calculation:diblock copolymer in a sphere

cylinder_block_uin.udf / uot.udf

Diblock copolymer in cylindrical mesh.

Chapter 5

Operation guide of SUSHI

5.1 SUSHI

SUSHI can be started by typing a command from the console. The supported operating systems are MS Windows and Linux. Although SUSHI has no implementation of GUI, one can use SUSHI as if it is running on a GUI by using the UDF format files on GOURMET.

5.2 File System of SUSHI

The directory structure of SUSHI is as follows.

```
OCTA/PF_ENGINE/SUSHI10.54/
       Susi--+--def_udf--+--SUSHIInput.udf     : definition of input UDF format
             |           +--SUSHIOutput.udf    : definition of output UDF format
             |           +--SUSHIParameter.udf : definition of parameter UDF format
             |
             +--include : include files
             |
             +--sample---+--Input      : sample inputs           ( sec. 5.5-7 )
             |           +--Input_V2   : sample inputs version 2 ( sec. 5.12 )
             |           +--Output     : sample outputs          ( sec. 5.8-10 )
             |           +--Seed_Input : sample inputs by SEED   ( sec. 5.14-15 )
             |
             +--src     : source files
```

"SUSHIInput.udf" contains definitions of the input UDF format, and is used when an input UDF file is created. On the other hand, "SUSHIOutput.udf" is usually not necessary. You can use it to know the data structure of the output UDF file.

The executable load modules are stored in the following directories.

```
OCTA/PF_ENGINE/SUSHI10.54/
Susi/bin----+--cygwin--------+--sushi.exe : load module for Cygwin
            |
            +--k-------------+--sushi : load module for K-computer(compile option)
            |
            +--linux---------+--sushi : load module for 32 bit Linux
            |
            +--linux_64------+--sushi : load module for 64 bit Linux
            |
            +--aix-----------+--sushi : load module for AIX(compile option)
            |
            +--win32---------+--sushi.exe : load module for 32 bit windows
            |
            +--x64-----------+--sushi.exe : load module for 64 but windows
```

5.3 Starting method

The current version of SUSHI can read/write files with two different formats, i.e. the UDF format and the SEED (Simple and Easy Editable Data) format. The UDF format files can be read or edited using the GUI of GOURMET. The UDF files have well-defined structures and therefore prevent the user from making mistakes when preparing the input files. Please refer to UDF manual about the details of the UDF format. On the other hand, the SEED format has a simple structure. Thus, it is easy for the user to prepare the SEED files although he/she has to type the keywords by himself/herself. The file of SEED format should obey the UNIX-type format (each line ends with LF).

Parllel version names are used as follows sushiPTL: Pthread Library using shared memory version. sushiGPU: NVIDIA GPU version. sushiMPI: MPI version.

Starting SUSHI from the console

To start SUSHI from the console, the following command should be used.

```
> sushi -I full_input_file_name
        -O full_output_file_name
        -R full_restart_file_name -rrecord_number_for_restart [-w]
        -L scf_log_file_name_of_dynamics
        -M message_file_name
        -C calculation_profile_file_name
        -A archives_of_latest_record_file_name
        -S summary_of_calculation_file_name
        -P control_parameter_file_name
        -Z special_COGNAC_input_file_name
        -n number_of_core_for_pthread_calculation
        -b number_of_threads_per_block_for_CUDA
        -d parameter_for_GPU_device
```

Here the arguments -I, -O, -R, -L, -M, -C, -A, -S, -P, -Z are parameters specifying the names of input/output and supplementary files. The meanings of these arguments of SUSHI are as follows

---

```
Input          -I full_input_file_name (including the file name extension)
                  In case this is a UDF format file, the file name extension
                  must be ".udf", and the data structure should match the
                  data structure defined in "SUSHIInput.udf" file.
                  SUSHI automatically regards a file as an input file, if the
                  file name ends with "_uin.udf". Therefore, the filename of an
                  input UDF file is in general "input_file_name_uin.udf".
                  In case the input file is a SEED format file, the file name
                  extension must be ".sin", and therefore the general file name
                  of a SEED input file becomes "input_file_name.sin".
Output         -O full_output_file_name (including the file name extension)
                  The simulation data are written in this file with the data
                  format defined in "SUSHIOutput.udf".  When this parameter
                  is not specified, SUSHI automatically creates an output file
                  with the name "input_file_name_uot.udf", where "input_file_name"
                  is the same as that of the input file name specified above.
Restart        -R full_restart_file_name (including the file name extension)
                  This parameter specifies an output file name of a previous job
                  whose data are used for the initial condition of the present job.
                  The data structure of this restart file must obey those defined
                  in "SUSHIOutput.udf" file. When this parameter is not specified,
                  SUSHI assumes "input_file_name_uot.udf" as the restart file name
                  and appends new record to the end of this file.
```

```
                -r record_number_for_restart
                   This parameter specifies which data in "full_restart_file_name" file
                   are used as the initial condition for the present restart job.
                   The "record_number_of_restart" parameter corresponds to the
                   \begin{record}{"Step number"} written at the top of each output record
                   in the "full_restart_file_name" file.
                   In case of a dynamic calculation, the final state of the system is
                   written in the "archive file" whose default name is
                   "input_file_name_uar.udf".  This archive file can conveniently be
                   used as the restart file, with which the restarting process becomes
                   faster.
                   If you do not use this option, SUSHI restarts from the final record.
                -w
                   Restart without the parsing procedure of the restart file.
                   This option is valid only for the files generated by SUSHI.
                   This option saves memory, accelerates to read and  will be efficient
                   when restarting using a large file.
                   As GOURMET modifies the output file of SUSHI and stores the result
                   to a file in a text format, it is necessary to parse the text
                   file when reading the file. We recommend to use this option.
Log of SCF      -L scf_log_file_name_of_dynamics
                   The status of the convergence scheme at every step of the SCF
                   iterations is written in this file. This file is overwritten at every
                   time step of the dynamic simulation.
                   Sometimes this file becomes considerably large. In such a case, the
                   output to this file can be suppressed by specifying the name
                   "/dev/null" (for the UNIX type OS) or "nul" (for Windows type OS).
                   If this parameter is not specified, SUSHI creates a file named
                   "input_file_name_usl.udf" automatically.
                   ( It has a extension udf but is not a UDF format file. )
Log             -C calculation_profile_file_name
                   The status of the time evolution of a dynamic simulation is written
                   in this file. The status of the SCF convergence is written in this
                   file in the case of static calculation.
                   The interval between the consecutive file output events can be
                   specified by the SCF_parameter.output_interval_step or
                   the dynamics_parameter.output_interval_step in the input file.
                   When this Log file name is not specified, the standard output is used.
Archives        -A archives_of_latest_record_file_name
                   At every n time steps, the state of the system is written in this
                   archives file.
                   The time step interval n can be specified by the
                   dynamics_parameter.archives_output_interval_step in the input file.
                   When this archives file name is not specified, SUSHI creates a
                   file named "input_file_name_uar.udf" automatically.
Message         -M message_file_name
                   A control message ( RESUME, RUN, STOP, etc. ) is read from this file.
Summary         -S summary_of_calculation_file_name
                   SUSHI writes this file as a summary for GOURMET.
                   The max error profile for the judgement of the SCF convergence is
                   written to this file in the static equilibrium calculation.
                   The free energy profile is written to this file in the calculation
                   of dynamics. The number of record is limited to 100.
                   When this file name is not specified, SUSHI writes no summary.
                   Please refer to the GOURMET manual.
maximum number  -m maximum_number_of_summary_data
of summary data    This parameter specifies the maximum number of summary data which
```

```
                    means the size of horizontal axis of summary graph such as fig. 5.2.
                    The default value of the maximum number is 100.
 number of CPU  -n number_of_CPU
                    SUSHI uses multi threads of which the numner of threads is the same of
                    the number_of_CPU.
                    The compelation with "MULTICPUT=on" is needed for this option.
                    Attention, since the efficiency of parallel processing is low yet,
                    the parallel calculation is available for large systems.
 number of      -b number_of_threads_per_block_for_CUDA
threads per block   Number of threads per block for CUDA (default 256)
 parameter for  -d parameter_for_GPU_device
                    [order of device id to use].[coefficient for memory]
                    Example:"012.9" mesans that use device 0, 1, and 2 in the order of
                    device id, and use maximum memory unitil "device memory"*0.9
                    in eachi device.
                    All selected memories are used in the last device.
                    The memory of the final device is used over the "device memory"*0.9
                    with warning until the job is aborted.
                    8: When MPI version, use all devices(default only use id 0)
                    17: Show all devices name and terminate.
 Parameter      -P control_parameter_file_name
                    SUSHI reads control parameters from this file.
                    When this file name is not specified, No control parameter can be read.
```

Refer to Section 5.12

```
 Kill           -K kill_file_name
                    If you use this option, SUSHI will be killed by recognizing the
                    existence of the kill_file on the directory running SUSHI.
 InputUDF       -i
                    If this parameter is specified, SUSHI writes the definition of the
                    UDF input data structures defined in "SUSHIInput.udf" file to the
                    standard output and SUSHI terminates.
 OutputUDF      -o
                    If this parameter is specified, SUSHI writes the definition of the
                    UDF output data structures defined in "SUSHIOutput.udf" file to the
                    standard output and SUSHI terminates.
 SEEDinput      -s
                    If this parameter is specified, SUSHI sorts the data in the input
                    SEED file "input_file_name.sin" in the alphabetic order with respect
                    to the keywords. The sorted data are written in the standard output
                    and SUSHI terminates.
 Staging        -t mode
                    Staging means the gather and scatter operations of global and local files.
                    This is used  mainly for MPI parllel calculation because very large
                    global data can not keep on the rank0 process memory thus a global (data)
                    file must be separated to local (data) files.
                    You can invoke several options with mode as follows.
                    ------------------------------------------------------------------------
                    mode
                      0  Only output log file, for test work.
                      1  General staging, not use global file.
                      2  A global file is scattered to local files.
                      3  Local files are gatherd to a global file.
                    ------------------------------------------------------------------------
                    Where local files are written with number of rank as
                    000000,000001,000002......
                    You can use other options, please refer the description with -h option.
```

```
                  Attention:Text input file is useful for large scale calculation, thus
                  SEED format should be used for the calculation with staging.
SEEDconverter  -u
                  If this parameter is specified, SUSHI reads the input SEED file
                  "input_file_name.sin", converts it into a UDF format and store the
                  UDF data into a UDF file "input_file_name_uin.udf" and terminates.
version        -v
                  Show the version of SUSHI and terminates.
help           -h
                  Show the usage of SUSHI and terminates.
Zooming        -Z special_COGNAC_input_file_name
                  The special file for COGNAC, which is the "relax_in.udf" or "restart_in.uid"
                  in def_udf for zooming.
                  When the relax_in.udf is used, the coordination of beads for COGNAC are genera
                  by COGNAC.
                  When the restart_in.udf is used, the coordination of beads for COGNAC are gene
                  by SUSHI.
                  New file relax_input_file_in.udf or relax_restart_file_in.udf for COGNAC
                  will generated.
```

If a file with the SEED format is specified as the input file of SUSHI, SUSHI creates corresponding input file with the UDF format named

"input_file_name_uin.udf" (SUSHIInput.udf format),

"input_file_name_uinv2.udf" (SUSHIInputV2.udf format), and

"input_file_name_upr.udf" (SUSHIParameter.udf format) automatically.

Execution using MPI

Run sushiMPI as

```
> mpiexec/mpirun -n number_of_processes sushiMPI -I.....
```

where selection of MPI command, mpiexec or mpirun, is depended on the MPI library you use when compelation.

The number_of_processes must match the product of number of division of all axes for MPI. All axes must be devided to use MPI. Please refer the input method of Mesh UDF.

Starting SUSHI from GOURMET

In GOURMET, we use the terminology "Engine" for the simulators. The same terminology is also used in UDF for the same purpose. Thus, in the following, the word "engine" always means a simulator.

- Start GOURMET and select "Tool/Engine Run". Then, you can see the "Engine Run" window as follows.
- In the "Run name:" box, specify the desired run name.
- In the "Engine:" box, specify the name of the suitable version of SUSHI for your operating system.
- In the "Input UDF:" and "Output UDF:" boxes, specify the names of the input UDF file and the output UDF file, respectively.
- Start the engine by clicking the "Run" button.
- After finishing your job, close the "Engine Run" window by clicking the "OK" button.

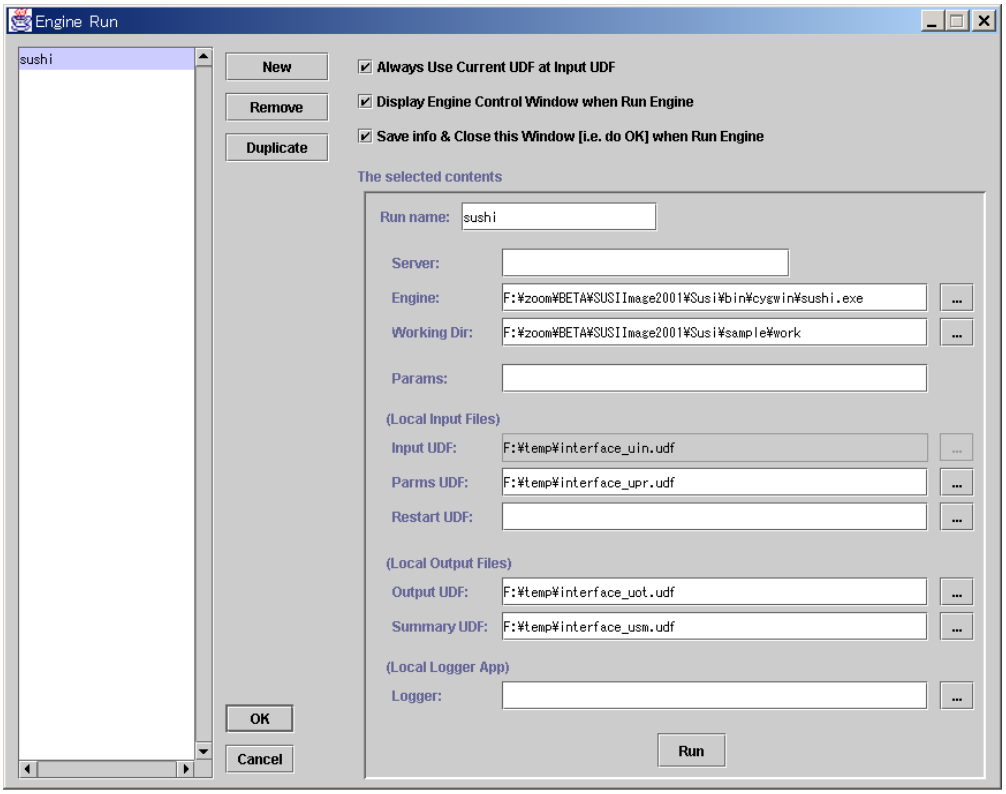


Figure 5.1: Engine run window

The UDF names that must be specified are Input(default) UDF and Output UDF. If you specify Summary UDF, you can check the change of the maximum error in the case of the static equilibrium calculation or the change of the free energy in the case of the dynamic calculation. If you specify Params UDF, you can control the execution of SUSHI. The next figure is the "Engine Control" window which is opened when "Run" button is clicked.

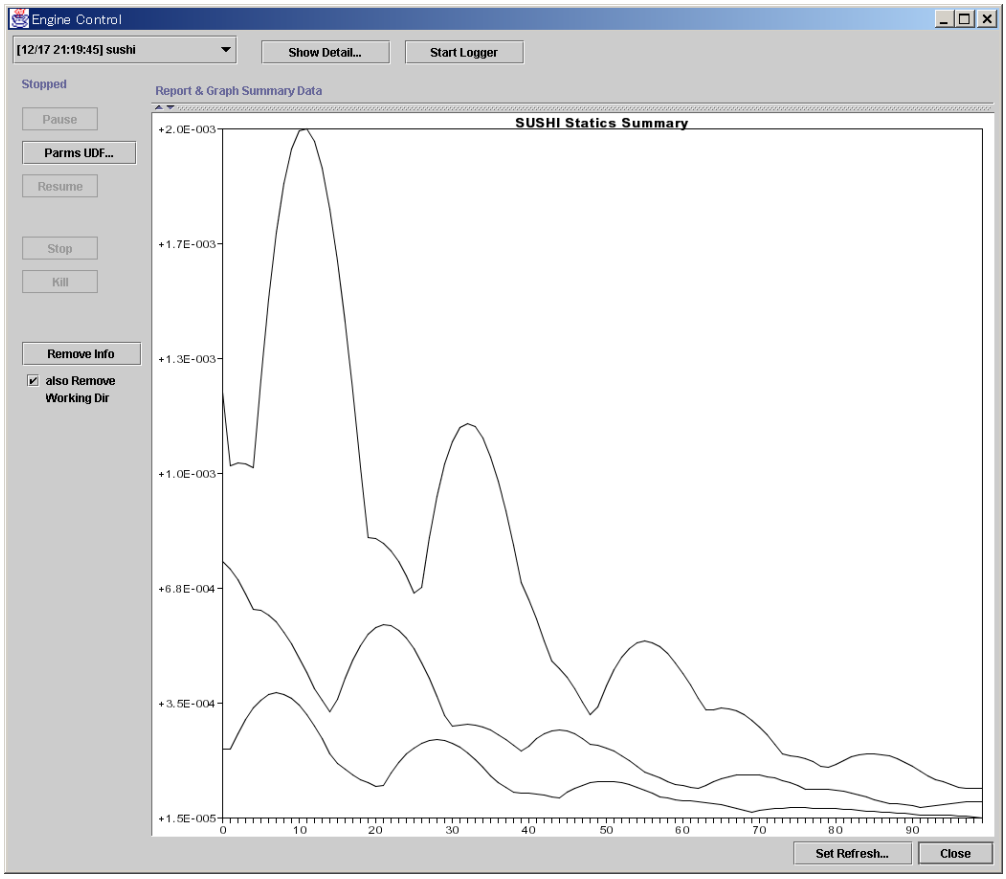


Figure 5.2: Engine control window

An example of the change of the maximum error is shown in the figure for the case of a static equilibrium calculation. You can change the size of horizontal axis of the graph by using -m option. When you click “Pasue” button, the execution of SUSHI pauses. While pausing, you can change the Parameter UDF file that stores the control parameters to open new GOUMET window by clicking the “Param UDF” button. After changing the Parameter UDF file, you can resume SUSHI by clicking “Resume” button. The definition of Parameter UDF is described in sec. 5.12. Clicking “Stop” button stops the execution of SUSHI by sending a messag to it. Clicking “Kill” button kills the running process of SUSHI.

For the details, readers should refer to the “GOURMET Operation Manual”.

5.4 Terminating SUSHI

There are two ways to terminate SUSHI. One way is to use the control window on GOURMET mentioned above. The other is to put a file with arbitrary contents but with the name ”input_file_name.stp” on the same directory that the SUSHI is running. In the case of the static equilibrium calculation, the job will be terminated when an SCF iteration step is finished. In the case of the dynamic calculation, SUSHI terminates when one time step of the dynamics is executed. Before terminating, SUSHI writes the current status of the system to the output file.

5.5 Input UDF header

The header part of an input UDF file expresses the type of the engine, and has the following form.

```
\begin{header}
\begin{def}
        EngineType:string;
```

```
        EngineVersion:string;
        IOType:string;
        ProjectName:string;
\end{def}
\begin{data}
        EngineType:"SCFEngine";  // Engine type (reserved keyword)
        EngineVersion:"141205";  // Engine version (reserved keyword)
        IOType:"IN";             // Input-and-output type (always "IN" for input UDF files)
        ProjectName:"WG2";       // Project name (reserved keyword)
\end{data}
\end{header}
```

In the current version of SUSHI, the following keywords should always be used.

```
EngineType:     "SUSHI";
```

EngineVersion: ” 160104”;

```
IOType:         "IN";
```

5.6 Input UDF definition

The data structures in the input UDF files for SUSHI take the following form, where the sentences starting with // are comments. Although default values are defined for several parameters, we recommend the users to specify all the parameters explicitly so that the conditions of the simulation system become clear. After reading this section, please refer to Appendix A which explains the SUSHIInputV2.udf format. The format is the upper version of the SUSHIInput.udf.

```
SUSHIInput:{
  // Parameters for controlling the calculations ################################
    calculation_method:CalculationMethod // The parameter for calculation method.
    start_condition:select { "START", "CONTINUE", "RESTART", "RESTART_READMESH" }
        // The flag for starting condition
        // START            : normal start
        // CONTINUE         : continue with reading the mesh at the final recoerd
        // RESTART          : restart without reading the mesh at the final recoerd
        // RESTART_READMESH : restart with reading the mesh at the final recoerd
    solver_parameter:SolverParameter // The parameter for solver.
  // Mesh and boundary condition ################################################
    mesh:Mesh
    type_of_free_propagator_of_regular_mesh:select
                                          { "1NN-P","2NN-NP","2NN-P","3NN-P" }
        // The type of the discretized Laplacian operator.
        // This is for the regular mesh only.
    boundary_condition:BoundaryCondition

        // Refer to Section 2.9.3.

  // Polymer and solvent ########################################################
    monomers[]:Monomer
        // Array of a monomer.
    monomer_SCF_character_table[]:MonomerSCFChar
        // Array of the characteristic properties of a monomer.
        // This parameter is required only when the monomer has the internal
        // states, for example, a tapered structure etc.

        // Refer to Section 2.9.3.

    components:Components
        // A set of components (polymers and solvents) that can be used in the
```

```
        // simulations.
        // ( Caution )
        //   ID's of the polymers and the solvents used in the following are
        //   the index numbers of components.polymers[] and components.solvents[].
    volume_fractions: VolumeFractions
        // The volume fractions of the components in the system.
        // ( Caution )
        //   Only the components listed in volume_fractions.polymer_volume_fractions[]
        //   and volume_fractions.solvent_volume_fractions[] are used in the simulations.
    chi_parameters[]:ChiParameter
        // Array of Chi parameter.
  // Physical quantities that should be output. ################################
    properties:Properties
        // Specify the physical quantities that should be output.
  // External conditions ########################################################
    external_conditions:ExternalConditions
        // External conditions to the system.
}
```

5.6.1 Main classes in the SUSHIInput

SUSHIInput.calculation_method

```
class CalculationMethod:{
    type:select { "STATICS", "DYNAMICS", "MONTECARLO" }
    DYNAMICS:DynamicsParameter
    MONTECARLO:MonteCarloParameter
}
```

SUSHIInput.calculation_method.DYNAMICS

```
class DynamicsParameter:{ // These parameters are valid when the type is DYNAMICS.
    delta_t:double                // The time mesh width.
    variable_delta_t:{//  class VariableDelt   Parameters for time mesh method
        max_delta_t:double        //  The maximum delta_t
        variable_coef:double      //  The coefficient for delta_t
                                  //  Procedure;  //---------------------
                                  //  Start a dynamics
                                  //  (1) Calculate segment densities
                                  //  If all segment densities > 0
                                  //    delta_t = variable_coef * delta_t
                                  //    If delta_t > max_delta_t then
                                  //        delta_t = max_delta_t
                                  //    goto (2)
                                  //  else
                                  //    delta_t = delta_t / variable_coef
                                  //    goto (1)
                                  //  (2)  goto next time step
    }
    max_dynamics_step:int         // The total numbers of the time steps.
    output_interval_step:int      // Output step interval.
    archives_interval_step:int    // The output step interval to archives file.
                                  // The default value is the same as output_interval_step.
    log_interval_step:int         // The output step interval for log.
```

```
                                // The default value is 1.
    dynamics_scheme:select { "EXPLICIT" , "EXPLICIT2" , "IMPLICIT" }
            // The default value is EXPLICIT.
            // The scheme for the integration of the equation of motion.
            // EXPLICIT // The explicit scheme. Usually, this scheme is used. Euler scheme.
            // EXPLICIT2// The2step Runge-Kutta scheme.
            // IMPLICIT // The implicit scheme.
    compressibility:double      // Compressibility.
                                // Complete imcompressible condition is available by 0.
}
```

SUSHIInput.calculation_method.MONTECARLO

It is need to input the parameters to SUSHIInput.external_conditoins.monte_carlo_conditions for the Monte Carlo simulation. Refer to 2.9.17 and 5.7.10.

```
class MonteCarloParameter:{ // These parameters are valid when the type is MONTECARLO.
    max_monte_carlo_step:int    // The total numbers of the Monte Carlo steps.
    output_interval_step:int    // Output step interval.
    archives_interval_step:int  // The output step interval to archives file.
                                // The default value is the same as output_interval_step.
    log_interval_step:int       // The output step interval for log.
                                // The default value is 1.
}
```

SUSHIInput.solver_parameter

```
class SolverParameter:{ // The ‘‘solver" means a special-purpose simulator/simulation method.
    type:select { "ADF", "FH", "RPA", "SCF" }
    SCF:SCFParameter // The SCF (Self Consistent Field) method.
    RPA:RPAParameter // RPA (Ginzburg-Landau using Random Phase Approximation)
    ADF:ADFParameter // The ADF (Approximate Density Functional) method.
                     // (Under construction)
    FH:FHParameter   // The Cahn-Hilliard type dynamics method.
                     // The sample program, Refer to Appendix.
}
```

SUSHIInput.solver_parameter.SCF

```
class SCFParameter:{
    delta_s:double              // The mesh width for the chain length used in the
                                // calculation of the path integral.
                                // This parameter is positive and the same value is used
                                // in the calculations of the path integral for both
                                // directions along the chain.
    constV:double               // The constant parameter used in the updating of
                                // the chemical potential in the SCF iteration scheme.
    constW:double               // The constant parameter used in the updating of
                                // the interaction between segments in the SCF
                                // iteration scheme.
    error:double                // The threshold value used in the judgement of the
                                // convergence of the iterations.
    random_seed:int             // Seed for the random number generator.
```

```
    standard_deviation:double     // The standard deviation of the Gaussian random numbers
                                  // used as the initial values of the segment density
                                  // fields.
    judge_method:select { "ABSOLUTE", "RELATIVE" }
                                  // The method to judge the convergence of the iterations.
                                  // This parameter is used only in the dynamic SCF
                                  // calculations.
        // ABSOLUTE : When the absolute value of the change in the density field "phi"
        //            becomes less than "error", the iteration scheme is terminated.
        // RELATIVE : When the relative value of the change in the density field "phi"
        //            becomes less than "error", the iteration scheme is terminated.
        //            The RELATIVE method is usually severer than the ABSOLUTE method.
    convergence_test_interval_step:int  // Interval of above test. 0 means every step(i.e. = 1).
    max_SCF_step:int              // The maximum number of iterations allowed for the
                                  // SCF calculation.
    output_interval_step:int      // During the SCF iterations, the data are written
                                  // into the file/screen at every other
                                  // "output_interval_step"  iteration steps.
    SCF_method:select { "INCORE", "DIRECT" }
        // How to handle the memory area for the large data of the path integral.
        // INCORE : All the data of the path integrals are kept on the memory.
        //          In this case, the calculation is faster but the system requires more
        //          memories.
        // DIRECT : The data of the path integrals are recalculated when they are required.
        //          In this case, the calculation is slower but the system requires
        //          less memories.
    pathintegral_scheme:string    // The type of the scheme with which the evolution
                                  // equation for the path integrals is solved.
        // explicit : The explicit scheme. With a proper choice of "delta_s", this method
        //            can usually be used.
        // implicit : The implicit scheme. When the integration of the path integral
        //            equations diverges, this method may improve the convergence of
        //            the integration.
}
```

SUSHIInput.solver_parameter.RPA

Parameters for GRPA, refer to Section 2.6.

```
class RPAParameter:{
    random_seed:int               // Seed for the random number generator.
    standard_deviation:double     // The standard deviation of the Gaussian random numbers
                                  // used as the initial values of the segment density
                                  // fields.
        // Attention;   GRPA is available only for dynamics calculations.
        //       For the stable calculations
        //          The mobility type should be selected as ROUSE or REPTATION,

        //          Refer to Section 5.7.9.

        //          Time mesh method is recommended,
        //          i.e., variable_delta_t should be used.
}
```

SUSHIInput.mesh

```
class Mesh:{
    name:KEY
    type:select { "REGULAR", "RECTANGULAR", "CYLINDRICAL", "SPHERICAL" }
    axes[]:MeshAxis                // Array of mesh axes.
        // Array of mesh axes.
        // Its dimension is the total number of coordinate axes of the system.
    index_rule[]:int
        // This specifies how the array elements (i, j, k) are arranged on the memory.
        // The first argument i runs first for example (0,0,0),(1,0,0)..(X-1,Y-1,Z-1).
        // For SUSHI, this is fixed as [0, 1, 2] except cylindrical mesh and
        // is fixed as [2, 1, 0] for cylindrical mesh.
        // Although this parameter does not affect the functions of SUSHI,
        // it is used when the mesh data is passed to another simulator and
        // is passed to the viewer on GOURMET.
}
```

SUSHIInput.boundary_condition

```
class VolumeFractionsOnBoundaries:{
    polymer_volume_fractions[]:VolumeFractionOnBoundary
    solvent_volume_fractions[]:VolumeFractionOnBoundary
}
class BoundaryCondition:{
    conditions[]:AxisBoundaryCondition  // An array of the boundary conditions for each
                                        // axis.
    volume_fractions_on_boundaries:VolumeFractionsOnBoundaries
}
```

SUSHIInput.components

```
class Components:{
    polymers[]:Polymer // Array of polymers.
    solvents[]:Solvent // Array of solvents.
}
```

SUSHIInput.properties

```
class Properties:{
    segment_volume_fraction_conditions[]:SegmentVolumeFractionCondition
        // Array of the volume fractions of each segment species.
    radius_of_gyration_conditions[]:SubchainUnit
        // Array of radius of gyration (Rg) conditions.
        // This parameter specifies the subchain for which the gyration radius is
        // calculated.
    scattering_function:ScatteringFunctionInput
       // flag for scattering function calculation

}
```

SUSHIInput.external_conditions

```
class ExternalConditions:{
    surface_chi_parameters[]:SurfaceChiParameter
        // Array of the chi-parameters for the interaction between the wall
        // and the segments.
    graft_conditions[]:GraftCondition
        // Array of the grafting conditions. Graft the free ends on the sold surface.
    mask_conditions[]:MaskCondition
        // Array of the mask conditions. Constrain the region where the free end
        // can exist.
    static_conditions:StaticConditions
        // The effective conditions for static equilibrium calculations.
    dynamic_conditions:DynamicConditions
        // The effective conditions for dynamic equilibrium calculations.
    monte_carlo_conditions:MonteCarloConditions
        // The parameters for Monte Carlo calculation.
    electrostatic_conditions:ElectrostaticConditions
        // The parameters for strong polyelectrolyte.
    obstacles:Obstacles
        // The parameters for obstacles.
}
class StaticConditions:{
    polydispersity_conditions[]:PolydispersityCondition
        // Array of the conditions on the polydisperse homopolymer systems for which
        // the path integral for the longest chain is shared with the others.
        // This option saves the computational cost of the static calculation
        // for polydisperse linear homopolymer systems.

        // Refer sec. 2.9.12.

    symmetry_conditions[]:SymmetryCondition
        // Array of the definitions of the commonly used path integrals.
        // This option saves the computational cost of the static calculation
        // for polydisperse system which includes polymers with similar structures.
        // Refer sec. 2.7.12.
    domain_specification_conditions[]:DomainSpecificationCondition
        // Array of the conditions for setting the initial values of the
        // self-consistent field.
    constraint_conditions[]:ConstraintCondition
        // Array of the constraints used in the SCF calculations.
    system_size_optimization:SystemSizeOptimaization
        // Parameters for system size optimization.
}
class DynamicConditions:{
    segment_mobilities[]:SegmentMobility
        // Array of the mobilities of each segment species.

        // Refer sec. 2.8 and 2.9.10.

    polymer_mobilities[]:LocalMobility
        // Array of the mobilities that depend on the volume fraction of the polymer.
    solvent_mobilities[]:LocalMobility
        // Array of the mobilities that depend on the volume fraction of the solvent.
    reaction_conditions_of_rapid_reactions[]:ReactionConditionOfRapidReaction
        // Array of the conditions for the reactions with fast reaction rates.
        // Refer to Fast reactions of sec. 2.7.15.
    reaction_conditions_of_active_sites[]:ReactionConditionOfActiveSites
        // Array of the conditions for the reactions with changes in the molecular
```

```
        // structures. Refer to Reactions with active monomers or with active sites
        // of polymers of sec. 2.7.15.
    reaction_conditions_of_grafts[]:ReactionConditionOfGraft
        // Array of the conditions for the grafting reactions.
```

// Refer to Grafting reactions of sec. 2.9.15.

```
    shear:Shear
        // Data structure of shear
    noise:Noise
        // Data structure of thermal noise
    system_size_dynamics:SystemSizeDynamics
        // Data structure of dynamical system size optimization
    hydrodynamics_parameters:HydrodynamicsParameters
        // Data structure of hydrodynamics
    hybrid:Hybrid
        // Data structure of hybrid method
```

5.7 Details of the definitions of the input UDF

5.7.1 Mesh

Refer to Section 2.9.3.

Mesh axis

SUSHIInput.mesh.axes[]

```
MeshAxis:{
    values[]:double
        // In case of the regular, spherical, or cylindrical mesh,
        // this array contains the minimum value and the maximum value
        // of the range along the axis,
        // and the total number of divided cells along the axis, respectively.
        // Example: 0. 32. 64
        // In case of the regular and MPI calculation,
        // add the total number of divided cells for MPI along the edge.
        // Example: 0. 32. 64 2(for MPI)
        // In case of the rectangular mesh,
        // the array of the coordinates of the mesh points are stored.
}
```

Boundary conditions for each mesh axis

SUSHIInput.boundary_condition.conditions[]
Refer to Section 2.9.4.

```
AxisBoundaryCondition:{  // The boundary conditions at the both ends of the axis are
                         // specified.
    axis_conditions[]:string
        // PERIODIC :Periodic boundary conditions
        //           (In this case, specifying only axis\condition[0] is enough.).
        // DIRICHLET or WALL :Absorbing wall,
```

```
        //    i.e. the Dirichlet boundary condition with vanishing boundary
        //         for path integral.
        // NEUMANN :Reflective wall,
        //    i.e. the Neumann boundary conditions with vanishing gradient.
}
```

Restrictions on the volume fractions at boundary

SUSHIInput.boundary_condition.polymer_volume_fractions[]
SUSHIInput.boundary_condition.solvent_volume_fractions[]

Using this parameter, a dynamic calculation with quasi-grand canonical ensemble can be performed. In such a calculation, first an initial state for a dynamic simulation with the canonical ensemble is prepared. In the system, the boundary which is facing to the bulk reservoir should be treated as the Neumann bounary (reflective boundary), where the volume fraction of each component is constrained to the bulk value. At every time step of the dynamic simulation, SUSHI corrects the values of the volume fractions in the system so that their boundary values are fixed to the bulk values. How to specify the values of the volume fractions inside the system and in the bulk reservoir will be shown later. Readers should refer to Section 5.7.4 for the detail.

```
class VolumeFractionOnBoundary:{
    ID:int                  // ID of the component for which the volume fraction
                            // is specified on the boundary.
    boundary_name:select
        {"XMin","XMax","YMin","YMax","ZMin","ZMax","RMin","RMax","HMin","HMax"}
      // Name of the boundary on which the volume fraction
      // is fixed.
      // Xmin : YZ-plane with the minimum value of the X-axis.
      // XMax : YZ-plane with the maximum value of the X-axis.
      // YMin : ZX-plane with the minimum value of the Y-axis.
      // YMax : ZX-plane with the maximum value of the Y-axis.
      // ZMin : XY-plane with the minimum value of the Z-axis.
      // ZMax : XY-plane with the maximum value of the Z-axis.
      // RMin : The boundary surface with the minimum value of the R-axis.
      // RMax : The boundary surface with the maximum value of the R-axis.
    volume_fraction:double  // Values of the volume fractions on the boundary.
```

5.7.2 Definition of monomer

Readers should refer to Section 2.9.1 for the detail.

Monomer

SUSHIInput.monomers[]

```
Monomer:{
    species_name:string           // Name of the monomer.
        // If the monomer has its internal states, for example, the tapered block
        // copolymer, this name specifies the sequence in the monomer such as "taperAB".
    specific_volume:double        // Specific volume of the monomer.
        // This parameter is used when the path integral and the volume fraction of
        // the monomer are calculated.  If the monomer has its internal states, this
        // parameter is re-calculated using the specific volumes of each state, and will
        // be overwritten.
```

```
    effective_bond_length:double  // Effective bond length corresponding to a single
                                  // monomer.
        // This parameter is used for the calculation of the coefficient of the Laplacian
        // term in the evolution equation for the path integrals.

        // Refer to the formula Eq. ( 2.29).
}
```

Internal states of a sub-chain

SUSHIInput.monomer_SCF_character_table[]
If the monomer has its internal states, their characteristic properties are specified here.

```
MonomerSCFChar:{
    name:string    // Name.
        // A monomer with this name has to be defined in advance.
        // For example, a monomer that is composed of a tapered block of A and B monomers
        // may be named "taperedAB".
    states[]:State // Array of the internal states.
    // Array of the elements of the transition state probability matrix for the internal
    // states.
    array_of_transition_state_probabilities[]:TransitionStateProbabilities
}
```

State (Internal state of a subchain)

SUSHIInput.monomer_SCF_character_table[].states[]

```
State:{
    name:string         // Name of the state.
        // For example, in the case of a tapered polymer composed of "A" and "B" monomers,
        // "A" or "B" should be specified.  Either "A" or "B" has to be defined as
        // a Monomer in advance.
    probability:double  // The probability of finding this state in the subchain.
}
```

Transition state probability between internal states

SUSHIInput.monomer_SCF_character_table[].array_of_transition_state_probabilities[]

```
TransitionStateProbabilities:{
    probabilities[]:double // Array of the elements of the transition state probability
                           // matrix between the internal states of the subchain.
        // The transition state probability Tij is defined as the probability of finding
        // a monomer with state j next to a monomer with state i separated by the distance
        // equal to the mesh width along the chain.
        // In the current version of SUSHI, only the polymers with fixed concentration
        // distribution of segment kind along the chain can be calculated. Therefore, the
        // element Tij is determined only by the probability of finding the internal state
        // j at the specified position.
        // The position is specified by the distance from the junction with samller ID.
```

```
        // The value of the distance corresponds to the product of the index of this
        // data structure element in the array_of_transition_state_probabilities[] and
        // the mesh width along the chain( delta_s in SUSHIInput.solver_parameter.SCF ).
        // The total number of array elements should coincide with
        // "( subchain length / delta_s )+ 1".
}
```

5.7.3 Definitions of the polymer and solvent

The data structures "Polymer" and "Monomer" define a list of the components that can be used in the calculations. Among these components, only those that are listed in the "volume_fraction" parameter are actually used in the simulation. Readers should refer to Section 2.9.1.

Polymer

SUSHIInput.components.polymers[]

A polymer is defined as a set of connected subchains. A subchain is defined by its kind (species) and its length (number of monomers in the subchain). The topology of a polymer is specified by giving the ID's of the subchains and the ID's of the junctions on both ends of the subchain. For a few simple topologies, there is an easy way to specify them.

```
Polymer:{
    type:select {"HOMO","BLOCK","COMB","STAR","GENERAL" }
        // The keyword specifying the branching structure of the chain,
        // i.e. how to connect the subchains.
        // HOMO :  a homo polymer.
        // BLOCK : a linear multiblock copolymer made of a sequence of the subchains
        //         with the same order as they are stored in this array.
        // STAR : star block copolymer made of subchains which are connected at a single
        //        junction point.
        // COMB : comb-type block copolymer.
             // The order of the subchains stored in the array is as follows.
             // main chain - side chain - main chain - side chain -.....
             // For example, an array of the subchains A1, B1, A2, B2, and A3
             // indicates the following structure.
             //     A1--+--A2--+--A3
             //         B1     B2
        //  GENERAL : The way how the subchains are connected each other
             //       is specified by the ID's of the two junction points at the
             //       both ends of each subchain.
    blocks[]:Block                // Array of subchains.
    junction_pairs[]:JunctionPair // Array of the pair of the ID's of the junction points
             //                      at the both ends of the subchain.
             // This parameter is available only when the type of the polymer is GENERAL.
             // The values of the ID must start from 0.
             // For example, when the number of elements of Block is 1 and a pair of ID's
             // is [0, 0], it means that this polymer is a ring polymer.
}
Block:{         // Subchain.
    monomer_name:string
        // The name of the monomer which constitutes this subchain.
        // This monomer name must be defined as a Monomer in advance
    number_of_monomer:double
        // The total number of monomers contained in this subchain.
}
```

```
JunctionPair:{  // A pair of the ID's of the junction points at the both ends of
                // this subchain.
    first:int
    second:int
}
```

Solvent

SUSHIInput.components.solvents[]

```
Solvent:{
    name:string            // Name of the solvent.
    specific_volume:double // Specific volume of the solvent.
}
```

5.7.4 Volume fraction

Among the Polymer's and Solvent's defined in the previous sub-section, only those whose volume fractions are specified are used in the simulation. For different types of statistical ensembles, the meanings of the volume fractions are also different. Refer to Section 2.9.8.

Volume fractions of the components

SUSHIInput.volume_fractions

```
VolumeFractions:{
    polymer_volume_fractions[]:VolumeFraction  // Array of the volume fractions of
                                               // the polymers.
    solvent_volume_fractions[]:VolumeFraction  // Array of the volume fractions of
                                               // the solvents.
}
VolumeFraction:{  // Volume fraction.
    id:int // The element index of the component. The number starts from 0.
    volume_fraction:double
        // If the ensemble is CANONICAL, this parameter specifies the total volume
        // fraction in the system.
        // If the ensemble is GRANDCANONICAL, this parameter specifies the equilibrium
        // volume fraction in the bulk phase.
    ensemble:select { "CANONICAL", "GRANDCANONICAL" }
        // Statistical ensemble of the system.
        // CANONICAL : Canonical ensemble.
        //     Static calculation and Dynamic calculation are available for this case.
        // GRANDCANONICAL : Grand canonical ensemble.
        //     Static calculation is available for this case.
        //     Dynamic calculation is NOT available for this case.
    bulk_volume_fraction:double  // The volume fraction in the bulk phase.
        //     This parameter is used when the volume fractions in the simulation box and
        //     those in the reservoir should be specified separately.
        //     If you do not use this feature, set -1 to this parameter.
}
```

As was explained in the section **Restricted conditions of volume fraction on a boundary** of 5.7.1,

in the quasi grand canonical ensemble case, the values of the volume fractions in the system and in the bulk reservoir can be specified separately. In this quasi grand canonical case, the value of the parameter "ensemble" should be CANONICAL.

5.7.5 Interaction parameters between segments (χ parameters)

SUSHIInput.chi_parameters[]
Refer to Chapter 7.

```
ChiParameter:{
    name_i:string     // The name of the species of i-th monomer.
    name_j:string     // The name of the species of j-th monomer.
    parameter:double  // The value of Chi_ij.
                      // Chi_ji is automatically set using the symmetric relation
                      // Chi_ji =  Chi_ij.
                      // If the value is not defined, it is assumed to be 0.
}
```

5.7.6 Physical properties

SUSHIInput.properties
The user can obtain several physical quantities which are specified using the ID's of the polymers, solvents, subchains, and junctions, which correspond to their element indices in the arrays defined in the SUSHIInput.components. Here, as the rule in C language, the element index begins from 0.

```
class Properties:{
    segment_volume_fraction_conditions[]:SegmentVolumeFractionCondition
    radius_of_gyration_conditions[]:SubchainUnit
    scattering_function:ScatteringFunctionInput
}
```

Volume fractions of segments

SUSHIInput.properties.segment_volume_fraction_conditions[]
The volume fraction of a specified part of a subchain is calculated. This function is unavailable for polymers with a loop.

```
class SegmentVolumeFractionCondition:{
    polymer_ID:int      // ID of the polymer.
    subchain_ID:int     // ID of the subchain.
    begin_length:double // Position of a boundary of the section on the subchain for which
                        // the volume fraction is calculated.
                        // This length is the distance between the boundary and the
                        // junction with a smaller ID among the two junctions at the ends
                        // of the subchain.
                        // In order to define the ID's of the two junctions at both ends,
                        // the polymer should be defined as the GENERAL type.
                        // The length is measured in unit of "delat_s" in the
                        // "SCF_parameter".  This value should not exceed the length of
                        // the subchain.
    end_length:double   // Position of the other boundary of the section.
                        // This length is the distance between the boundary and the
```

```
                        // junction with a smaller ID among the two junctions at the
                        // ends of the subchain.
                        // When the length of the section is not an integer multiple of
                        // "delta_s", it is redefined as
                        // delta_s * int(section_length / delta_s).
}
```

Radius of gyration (R_g)

SUSHIInput.properties.radius_of_gyration_conditions[]
This parameter specifies the subchains whose radii of gyration are calculated and output.

```
classsubchainUnit:{
   polymer_ID:int      // ID of the polymer.
   subchain_ID:int     // ID of the subchain. -1:calculate Rg of full structure.
}
```

Scattering function

SUSHIInput.properties.scattering_function
Flag for output scattering functions

```
class ScatteringFunctionInput:{
    calculate:select { "ON", "OFF", "RESTART" }
                      //  ON   : calculate scattering functions
                      //  OFF : no calculate scattering functions
                     //  RESTART : calculate scattering functions of existing records

                        //            without date of the existing records
    intensity:select { "ON", "OFF" }
                      //  ON   : calculate multi-dimensional scattering function
                      //  OFF : no calculate multi-dimensional but 1D scattering function
    coef_for_mesh:double // The coefficient for mesh width of 1D scattering function.
                         // The mesh witdh = this value times 2 pi /(the longest length of edge).
}
```

5.7.7 External conditions

SUSHIInput.external_conditions
The user can set several external conditions which are specified using the ID's of the polymers, solvents, subchains, and junctions, which correspond to their element indices in the arrays defined in the SUSHIInput.components. Here, as the rule in C language, the element index begins from 0.

```
class ExternalConditions:{
    surface_chi_parameters[]:SurfaceChiParameter
    graft_conditions[]:GraftCondition
    mask_conditions[]:MaskCondition
    static_conditions:StaticConditions
    dynamic_conditions:DynamicConditions
    monte_carlo_conditions:MonteCarloConditions
    electric_conditions:ElectricConditions
```

```
    obstacles:Obstacles
}
```

Interaction parameters between the wall and the segments (χ_s)

SUSHIInput.external_conditions.surface_chi_parameters[]
Refer to Section 2.9.7.

```
SurfaceChiParameter:{
    boundary_name:select
        {"XMin","XMax","YMin","YMax","ZMin","ZMax","RMin","RMax","HMin","HMax"}
      // Definition of the surface with which the segments interact.
      // Xmin : YZ-plane with the minimum value of the X-axis.
      // XMax : YZ-plane with the maximum value of the X-axis.
      // YMin : ZX-plane with the minimum value of the Y-axis.
      // YMax : ZX-plane with the maximum value of the Y-axis.
      // ZMin : XY-plane with the minimum value of the Z-axis.
      // ZMax : XY-plane with the maximum value of the Z-axis.
      // RMin : The boundary surface with the minimum value of the R-axis.
      // RMax : The boundary surface with the maximum value of the R-axis.
      // Partcle : surface of particle
      // Fiber : surface of fiber
    target_name:string // The name of the target segment.
    parameter:double   // the value of the chi-parameter.
    id:int // ID of each lind of obstacles (available when choosing obstacles).
}
```

Graft conditions

SUSHIInput.external_conditions.graft_conditions[]
A grafted chain can be realized by imposing a constraint on the initial value of the path integral at the free end(s). Refer to Section 2.9.7.

```
GraftCondition:{
    boundary_name:string // The name of the wall onto which the chain is grafted.
                         // This is the same as those in
                         // "SurfaceChiParameter.boundary_name".
    polymer_ID:int       // ID of the polymer to be grafted.
    junction_ID:int      // ID of the free end of the polymer to be grafted.
    id:int // ID of each lind of obstacles (available when choosing obstacles).
}
```

Mask conditions

SUSHIInput.external_conditions.mask_conditions[]
This parameter specifies the region within which the position of a free end is confined. This function can be used in simulating micelles etc. Refer to Section 2.9.14.

```
MaskCondition:{
    polymer_ID:int            // ID of the polymer whose end should be confined.
    junction_ID:int           // ID of the junction that should be confined.
```

```
    mask_regions[]:AxisRegion // Array of the regions on the axes in which the end
                              // is confined.
}
AxisRegion:{ // Specify an interval on an axis.
    axis_name:select { "X", "Y", "Z", "R", "H" }
                     // Name of the axis: X, Y, Z, R or H.
    r_min:double     // Minimum value of the interval.
    r_max:double     // Maximum value of the interval.
                     // When Maximum = Minimum, it corresponds to constraining the
                     // chain end at a point.
}
```

5.7.8 Effective external conditions in static equilibrium calculation

Specification of the longest polymer for the static equilibrium calculation on polydisperse linear homopolymer systems

SUSHIInput.external_conditions.static_conditions.polydispersity_conditions[]

This parameter is valid only for the static equilibrium calculations on systems composed of polydisperse linear homopolymers with a single-state monomers. Specifying this parameter will reduce the memory and computational time. Refer to Section 2.9.12.

```
PolydispersityCondition:{
    longestHomoPolymerId:int    // ID of the longest linear homopolymer.
    targetHomoPolymerId:int     // ID of the polymer whose path integral should be calculated
                                // by using that of the longest polymer specified above.
}
```

Specification of the similar path integrals for the static equilibrium calculation

SUSHIInput.external_conditions.static_conditions.symmetry_conditions[]

Refer to Section 2.9.12. This parameter specifies the subchains whose path integrals can be shared each other. This parameter is valid only for the static equilibrium calculations. The computational time of the path integrals will be reduced using this function. For example, in the calculation of two diblock copolymer chains $A_{10}B_{10}$ and $A_{20}B_{20}$, the path integral starting from the free end of the A subchain of the longer chain $A_{20}B_{20}$ is the same as that of the shorter chain $A_{10}B_{10}$, and can be shared. On the other hand, since the initial values of the path integral starting from the junction points are different for $A_{10}B_{10}$ and $A_{20}B_{20}$, these cannot be shared.

```
SymmetryCondition:{
    parent_path:Path // The path of the subchain whose path integral is shared with
                     // the other subchains.
    child_path:Path  // The path of the subchain whose path integral is substituted
                     // by the above-defined parent path integral.
}
Path:{ // The path for the path integral.
    polymer_ID:int        // ID of the polymer.
   subchain_ID:int        // ID of the subchain.
    begin_junction_ID:int // ID of the junction from which the calculation of the path
                          // integral starts.
    end_junction_ID:int   // ID of the junction at which the calculation of the path
                          // integral ends.
}
```

Domain specification

SUSHIInput.external_conditions.static_conditions.domain_specification_conditions[]
In order to produce a desired morphology, an appropriate initial condition for the self consistent field can be prepared using this parameter. This parameter is valid for the static equilibrium calculations. Refer to Section 2.9.13.

```
DomainSpecificationCondition:{
    name:string                 // The name of the segment species.
    domain_regions[]:AxisRegion // The array of the intervals on the axes.
                                // This is the same as those used in the mask conditions.
}
```

Constraint conditions in the SCF calculation

SUSHIInput.external_conditions.static_conditions.constraint_conditions[]
This parameter specifies the constraint conditions imposed on the system when the SCF iteration is performed.

```
class SCFUnitIDSet:{  // The set of ID's of the part of the chain/solvent .
    type:int      // The type of the target part of the chain/solvent.
        // 0 :polymer
        // 1 :solvent
    IDSet[]:int   // The set of ID's.
        // A set of ID's of the component, subchain, and internal state of the target part.
        // In the case of a solvent, ID's after the subchain ID are not used.
        // In the monomer with a single state, the state ID is not used.
}
class ConstraintCondition:{
    target_ID:SCFUnitIDSet     // The set of ID's of the part of the chain/solvent on
                               // which the constraint is imposed.
    constraint_coondition:int  // The physical quantity on which the constraint is imposed.
        // 1  :The volume fraction.  This is the only one that is implemented in the
        //     current version of SUSHI.
}
```

System size optimization

SUSHIInput.external_conditions.static_conditions.system_size_optimization
Optimize the system size to minimize the free energy density. This method is available in SCF calculation.

```
class OptimizingAxes:{
    x:int // 1:optimize 0:not optimize
    y:int // 1:optimize 0:not optimize
    z:int // 1:optimize 0:not optimize
    // If you optimize x and y axes keeping the ratio, input as
    //   1 1 0.
    // If you optimize x and y axes independently, input as
    //   1 0 0
    //    0 1 0.
}
class SystemSizeOptimaization:{
    max_iteration_of_optimization:int // Maximun step number of SCF
```

```
    cubic_system:int  // If this value is 1, the system is recognized as cubic system.
    optimizing_axes[]:OptimizingAxes //  Flags for optimization
    delta_dL:double                  //  Relative delta L for the derivative of F.
    allowed_error:double             //  Relative allowed error in the optimization of L.
}
```

If the SCF calculation is converged or exceeded the max_iteration_of_optimization, the system size is optimized and the next SCF step is started. The optimization is converged when the absolute value of (variation of the length of edge / the length of edge) is less than $1e^{-4}$.

5.7.9 Effective external conditions in dynamimc calculation

Mobility for each segment species

SUSHIInput.external_conditions.dynamic_conditions.segment_mobilities[]
The users can specify the values of the mobility for each kind of monomer or solvent separately. If this parameter is not specified, the mobility is assumed to be 1.0. This parameter corresponds to the constant L_0 in Eqs.(2.64) and (2.65). Refer to Section 2.9.10.

```
class SegmentMobility:{  // Mobility of a segment.
    segment_name:string  // The name of the segment/solvent whose mobility is specified.
    mobility:double      // The value of the mobility.
}
```

Position-dependent mobilities

SUSHIInput.external_conditions.dynamic_conditions.polymer_mobilities[]
SUSHIInput.external_conditions.dynamic_conditions.solvent_mobilities[]
The mobility of a polymer in a dilute solution in general depends on the local concentration of the polymer and the species of the surrounding matrix. In the dilute limit, the mobility can be written as $L_0\phi$, where the parameter L_0 accounts for the dependence on the properties of the chain and those of the matrix solvent. Refer to Section 2.9.10.

ROUSE case: When the chain length is much longer than that of the matrix solvent and the polymer concentration is small enough (dilute limit). the mobility L is given by $L = L_0\phi$.

REPTATION case: When the chain length is long and is almost equal to that of the matrix molecules, the chains are strongly entangled. In such a case, the mobility is given by $L = (L_0/N)\phi$, where N is the length of the target chain. This REPTATION case is not valid for a solvent.

```
class LocalMobility:{  // The position dependent mobility.
    component_ID:int   // ID of the component.
    type:string  // Type of the model for the mobility
            // CONSTANT
            // ROUSE
            // REPTATION
}
```

Fast chemical reactions

SUSHIInput.external_conditions.dynamic_conditions.reaction_conditions_of_rapid_reactions[]
If the characteristic time scale of the chemical reaction is faster than the time mesh size used in the dynamic calculation, it is reasonable to assume that a certain amount of the reactants turns into the same amount of the product instantaneously. One such example is a radical polymerization of monomers that produces a polymer with a fixed polymerization index. The type of the chemical reactions that can be treated in the current version of SUSHI is $A + B + C + \cdots \to D$, i.e. several reactants react to yield a single

product. Refer to Section 2.9.15.

```
class ReactionConditionOfRapidReaction:{
    reactant_IDs[]:ReactantIDSet    // Set of the ID's of the reactants.
    product_ID:int                  // The ID of the product. Only a single component can
                                    // be specified as the product.
    volume_fractions_of_reactants_in_product[]:double
                                    // Array of the volume fractions of reactants in
                                    // a product corresponding to the ‘‘reactant_IDs".
                                    // The sum of the values for all the reactants should
                                    // be equal to unity. For example, the reaction from
                                    // reactants of polymer A and B in the same length to
                                    // a product of block copolymer AB, the data are 0.5
                                    // and 0.5.
    reaction_constant:double        // Reaction rate constant.
}
```

Reactions with structural change of the molecules

SUSHIInput.external_conditions.dynamic_conditions.reaction_conditions_of_active_sites[]
In this case, the reaction is defined as an event that a set of junctions of polymers, free ends of polymers, or monomers react to form a polymer with a different molecular structure. The numbers of the reactants are limited to two, and the product is limited to only one. Therefore, the reaction should be second order. Refer to Section 2.9.15.

```
class ReactionConditionOfActiveSites:{
    reactant_IDs[]:ReactantIDSet    // Set of the ID's of the reactants.  The numbers of
                                    // reactants are limited to two.
    polymer_ID_of_product:int       // The ID of the product polymer. Number of product is
                                    // limited to one.
    map_of_subchains[]subchainMap   // Map of ID's of subchains between reactants and product.
    reaction_constant:double        // Reaction constant.
}
```

Grafting reactions

SUSHIInput.external_conditions.dynamic_conditions.reaction_conditions_of_grafts[]
Refer to Section 2.9.15.

```
class ReactionConditionOfGraft:{
    graft_condition:GraftCondition  // The graft conditions on the reactant polymer.
                                    // This is the same as that defined in graft
                                    // conditions previously.
    polymer_ID_of_product:int       // The ID of the reactant polymer.
                                    // The number of reactant is limited to only one
                                    // species.
    map_of_subchains[]subchainMap   // Map of ID's of subchain between reactants and
                                    // product.
    reaction_constant:double        // Reaction rate constant.
}
```

Definitions of the common data for the reactions

SUSIInput.external_conditions.dynamic_conditions
reaction_conditions_of_rapid_reactions[].reactant_IDs[]
reaction_conditions_of_active_sites[].reactant_IDs[]
reaction_conditions_of_active_sites[].map_of_subchains[]
reaction_conditions_of_grafts[].map_of_subchains[]

```
class ReactantIDSet:{  // Set of the ID's of the reactants.
    type:int   // The type of the reactant:
        // 0 : polymer.
        // 1 : solvent.
    IDSet[]:int // Set of the ID'(s).
        // The array of the set of ID'(s).
        //   In case of the fast reaction, the datum is only the ID of the reactant in
        //   the component.

       //   Refer to Section 2.9.10 ( Fast reaction ).

        //   In case of the reactions involving structural changes of the molecules,
        //   the data are the ID of the reactant in the component and the ID of the
        //   junction to react.

       //   Refer to Section 2.9.10 ( Fast Reactions with active monomers

        //   or with activesites of polymers ). If the reactant is a solvent, the ID of
        //   the junction is not needed.
}
class SubchainMap:{  // A map of corresponding ID's of the subchains between the
                     // reactant and the product.
    reactant_ID:int              // ID of the reactant.
        // The ID is the element index of the array ``reactant_IDs[]".
        // This is NOT the index of the array of the component.
        // As we assume that the reactions with structural change of the molecules
        // are second order, this parameter is either 0 or 1.
    // The following ID's of the subchains define the correspondence between the
    // the subchain in the reactant and that in the product.
    subchain_ID_of_reactant:int  // The ID of the subchain in the reactant.
    subchain_ID_of_product:int   // The ID of the subchain in the product.
}
```

Sheared dynamics

SUSHIInput.external_conditions.dynamic_conditions.shear

Input the shear rare and period. Refer to Section. When the period is zero, one direction shear is added. 2.8.1.

```
class Shear:{
    shear_rate:double
    period:double
}
```

Thermal fluctuation

SUSHIInput.external_conditions.dynamic_conditions.noise
Input the standard deviation of thermal_noise of segment density. The thermal noise satisfies the fluctuation-dissipation relation eq. (2.41).

```
class Noise:{
    random_seed:int // Seed for the random number generator.
                    // If 0 is set, the value is randomly changed referring time.
                    // If a positive value is set, the value is fixed and is not changed
                    // thus the positive value is used when only debugging.
    standard_deviation_of_thermal_noise:double
}
```

Dynamical system size optimization

SUSHIInput.external_conditions.dynamic_conditions.system_size_dynamics
Optimize the system size in dynamics. Refer to Section 2.9.18.

```
class SystemSizeDynamics:{
    parameter_of_system_size_dynamics:double  //  The Q in eq. (2.103).
    compressibility:double                    //  Compressibility in eq. (2.103).
    interval_of_system_size_dynamics:int      //  Interval step of invoking eq. (2.103).
    optimizing_axes[]:OptimizingAxes //  Flags for optimization
                                     //  Refer class SystemSizeOptimaization
    delta_dL:double                  //  Relative delta L for the derivative of F.
    allowed_error:double             //  Relative allowed error in the optimization of L.
}
```

Hydrodynamic Effects

SUSHIInput.external_conditions.dynamic_conditions.hydrodynamics_parameters
Introduce hydrodynamic effects, refer to Section 2.8.5.

```
class Viscosity:{ // Parameter for viscosity
    name:string                                     // Segment name
    viscosity:double                                // Viscosity
}
class HydrodynamicsParameters:{ // 流体力学的パラメータ
    density:double                                  // Density
    viscosities[]:Viscosity                         // Array of viscosity
    neglect_convection_term:int // If neglect the convection term (1st term) in eq. (2.66)
                                                    // input 1
    dynamics_scheme:select { "EXPLICIT" , "EXPLICIT2" , "IMPLICIT" }
            // Calculation scheme
            // EXPLICIT // The explicit scheme. default. Euler scheme.
            // EXPLICIT2// The2step Runge-Kutta scheme.
            // IMPLICIT
    poisson_solver_parameter:PoissonSolverParameter
            // Parameter for Poisson Solver for Pressure,
            // The detail is written in the section of polyelectrolyte.
    noise:Noise
            // Parameter to add a thermal fluctuation to Navier-Stokes equation.
            // The data structure is in the same as those used in the thermal fluctuation.
}
```

Hybrid method

SUSHIInput.external_conditions.dynamic_conditions.hybrid
Parameters for hybrid method, refer to Section 2.8.4.

```
class Hybrid:{
    SCF_step:int //  Number of SCF iteration : standard = 1
    RPA_step:int //  Number of GRPA iteration
    free_energy_by:select { "RPA", "SCF" }
        //  Clculation method of the free energy
        //  RPA GRPA
        //  SCF SCF theory
        // Attention
        // Please refer the section of the SUSHIInput.solver\_parameter.RPA
}
```

5.7.10 Conditions for SCF Monte Carlo calculation

SUSHIInput.external_conditions.monte_carlo_conditions

```
class MonteCarloConditionOfJunction: {
    // The position of junction updated in the Monte Carlo procedures.
    junction_ID:int    // ID of the junction.
    position:Vector3D  // Position of the junction.
}
class MonteCarloConditionOfPolymer:{
    polymer_ID:int    // ID of the polymer.
    monte_carlo_conditions_of_junction[]:MonteCarloConditionOfJunction
}
class MonteCarloConditions:{
    monte_carlo_conditions_of_polymer[]:MonteCarloConditionOfPolymer
}
```

5.7.11 Electrostatic condition (Strong polyelectrolyte)

SUSHIInput.external_conditions.electric_conditions
This option is valid when the positive value is set to
SUSHIInput.external_conditions.electrostatic_conditions.electrostatic_condition.dielectric_constant_of_system.

```
class ElectrostaticConditions: {
    // Electrostatic conditions for polyelectrolyte.
    electrostatic_condition:ElectrostaticCondition
    poisson_solver_parameter:PoissonSolverParameter
}
class ElectrostaticCondition: {
    dielectric_constant_of_system:double // The dielectric constant of the system.
    electrostatic_parameters_of_segment[]:ElectrostaticParametersOfSegment
    electrolyte[]:Electrolyte
    external_electric_field:Vector3D

        // Refer to 2.8.6.
```

```
      // The value of the vector of the external electric field.
    parameter_of_external_electric_field_for_dynamics
   :ParameterOfExternalElectricFieldForDynamics
}
class ElectrostaticParametersOfSegment: {
    name:string    // The name of the segment.
    charge:double  // The charge of the segment per unit volume.
    dielectric_constant:double  // The dielectric constant of the segment
}
class ParameterOfExternalElectricFieldForDynamics:{
    direction:select { "X", "Y", "Z" }
    alpha:double

       // Refer to 2.8.6.

      // Do not use external_electric_field at the same time.
}
class PoissonSolverParameter: {
    // The parameters for ICCG method.
    omega:double         // The acceleration parameter.
    allowed_error:double // The allowed error for the electric field.
    max_iteration:int
}
```

Poisson equation is defined as

$$\nabla^2 U(\mathbf{r}) = -\rho(\mathbf{r})/\epsilon \tag{5.1}$$

where U is the electrostatic potential, ρ is the distribution of the total electric charge of the system, and ϵ is the dielectric constant. In SUSHI, the ∇^2 and the scalar field is treated as a matrix and a vector. Here, the matrix of ∇^2, $U(\mathbf{r})$, and $\rho(\mathbf{r})/\epsilon$ are written as $\mathbf{A}$, $\mathbf{u}$, and $\mathbf{q}$ respectively and the Poisson equation is regarded as a linear algebraic equations given by

$$\mathbf{Au} = -\mathbf{q}. \tag{5.2}$$

To solve the equation, The SOR method had been used in SUSHI and the iterative equation is

$$\mathbf{u^{(k+1)}} = \mathbf{u^{(k)}} - \omega\mathbf{D}(\mathbf{Au^{(k)}} + \mathbf{q}) \tag{5.3}$$

where ω is the acceleration parameter and the $\mathbf{D}$ is the matrix that the diagonal elements are $1/a_{i,i}$ and other elements are 0. The parameter "omega" used in the class PoissonSolverParameter means the ω.

The current version SUSHI uses ICCG method to solve the linear algebraic equations, and the ω is not used.

5.7.12 Obstacles

SUSHIInput.external_conditions.obstacles

Obstacles can be existed in systems, however nowadays the obstacles are restricted as particles or fibers. The region of an obstacle can set in inner side or outer side of the shell of each particle.

```
class Particle: {
    // Parameters for a particle as an obstacle.
    position_of_center:Vector3D // Center position of the particle.
    radius_of_particle:double   // Radius of the particle.
    region_of_obstacle:select { "IN", "OUT" } // Obstacle existing region,
      // Inner region or Outer region of the particle.
}
class Fiber: {
    // Parameters for a fiber as an obstacle.
position_of_end0:Vector3D // End position of the particle.
position_of_end1:Vector3D // End position of the particle.
```

```
radius_of_fiber:double    // Radius of the fiber and end caps.
end_cap:select { "YES", "NO" } // Cap ends YES or NO (default YES)
region_of_obstacle:select { "IN", "OUT" } // Obstacle existing region,
      // Inner region or Outer region of the fiber.
}
class Hexahedron: {
    // Hexahedron is defined by 2 spatial origins with 3 directional vectors.
    position_of_origin0:Vector3D // origin1
    direction_a0:Vector3D // vector1 from origin1
    direction_b0:Vector3D // vector2 from origin1
    direction_c0:Vector3D // vector3 from origin1
    position_of_origin1:Vector3D origin2
    direction_a1:Vector3D // vector1 from origin2
    direction_b1:Vector3D // vector2 from origin2
    direction_c1:Vector3D // vector3 from origin2
    coefs_of_surface_chi[]:double // the coefficient for chi_s.
    region_of_obstacle:select { "IN", "OUT" } // Obstacle existing region,
      // Inner region or Outer region of the hexahedron.
}
class ShapeByGrids: { // VER. 140808
    input_filepath:string // the file path name of a polygon data.
    region_of_obstacle:select { "IN", "OUT" } // Obstacle existing region,
      // Inner region or Outer region of the shape.
}
class Obstacles:{
    // Data set of obstacles
    ndiv_for_volume_calculation:int
      // Number of division of per mesh to get the boundary of obstacles.
      // About 1000 is recommended.
    particles[]:Particle          // array of particles.
    fibers[]:Fiber                // array of fibers.
    hexahedrons[]:Hexahedron      // array of hexahedrons.
    shape_by_grids[]:ShapeByGrids // array of shapes.
}
```

5.7.13 Zooming

SUSHIInput.zoom

This class is designed for transfers of the calucluation results of SUSHI to other engines. SUSHI10.54 only supports the zooming to Kremer-Grest coarse-grained MD model for COGNAC.

```
class COGNACZoomingParameter: {
    b_per_sigma:double // the number of baeds of Kremer-Grest model per 1 segment.
                       // Theoretical value = 1.6878 but it should be modified with
                       // the value of d_s. i.e. b/sigma should be b/(n*d_s) where
                       // n is any integer.
    density:double     // The density of Kremer-Grest model, default value = 0.85.
    by:select { "RUN", "RESTART" }
       // RUN: automatically calculate segment's density fields with normal run.
       // RESTART: only calculate segment's density fields with restart run.
}
class Zoom:{ // COMMON
    type:select { "COGNAC" } // choose "COGNAC" for zooming.
    COGNAC:COGNACZoomingParameter
}
```

This method uses the relax_in.udf file for COGNAC. Please refer 5.3 section and the reference [32].

5.8 Definition of the common UDF

At the top of the output UDF file created by SUSHI, "SUSHIInput" defined in the previous sections and the following common data are written. The main contents of the common data are the result of the pre-processing of "SUSHIInput".

5.8.1 Coordinates of mesh

```
class Vector3D: { // 3-dimensional position vector.
    x:double
    y:double
    z:double
}
MeshData: {  // Array of the coordinates of the mesh points.
    position[]:Vector3D
}
```

5.8.2 Result of the analyses on the compositions

The results of the analyses on the compositions of the system is given.

```
class VolumeFractionData: {
    name:string             // The name of the specified group of the monomers.
    volume_fraction:double  // The volume fraction of the specified group of the monomers.
}
class SCFUnitData: { // The minimum unit to divide the component for the SCF calculation.
    // A polymer can be divided into the subchains and a subchain can be divided into the
    // kinds of the monomer. For example, the subchain of the sequence "tapered AB" has
    // two kinds of the monomer, which are "A" and "B". Of course the kind of the
    // subchain of homopolymer A is only "A". We call the kind of the monomer as "state".
    // The SCF unit is defined as the monomers in a polymer, a subchain and a state,
    // therefore it has three indices to identify, which are the ID of
    // the polymer, the ID of the subchain, and the ID of the state. The ID of the state
    // should be started from 0 and is valid in the subchain. The SCF unit is applied to
    // a solvent. SUSHI can not treat a multi state solvent, therefore a solvent is
    // identified by only the ID of the solvent.
    name:string             // The name of the state.
    volume_fraction:double  // The volume fraction of the state.
    ID_set[]:int            // Set of ID's.
        // The structure is different depending on the type of the SCFUnit as follows.
        // In case of a subchain with the states, ID of the polymer, ID of the subchain,
        // and ID of the state.
        // In case of a subchain with a single state, ID of the polymer, and ID of the
        // subchain.
        // In case of a solvent, ID of the solvent.
}
class JunctionData: {  // The list of ID's of junctions.
    ID_set[]:int  // The ID's of the polymer and the junction.
        // ID of the polymer and ID of the junction.
}
```

```
Composition: {  // Results of the Analyses of the Compositions.
    species[]:VolumeFractionData  // The total volume fraction of the specified group of
        // the monomers. This is the array of the sum of volume fraction of the groups.
        // For example, the monomer "taperedAB" is treated as single species and the
        // monomer "A" is treated as single species too.
    states[]:VolumeFractionData   // The volume fractions of sum of the states.
        // The state means each of the internal states of a monomer.
        // For example, the "taperedAB" monomer has two states A and B.
        // If the system has no polymer with internal state, the species[] and the states[]        //
    SCF_units[]:SCFUnitData     // Volume fractions of the SCFUnits.
    junctions[]:JunctionData    // The list of all junctions.
}
```

5.9 Definitions of the output UDF

An output UDF file is composed of a definition part at the beginning, and a sequence of separated records. The beginning and the ending of a record are described as follows.

```
\begin{record}{"record number"}
\begin{data}
    $\sim$ data of the record.
\end{data}
\end{recoed}
```

The "record number" is written at the beginning of each record. This "record number" is the same as that used in GOURMET. Note that this "record number" is different from the following "step number" in the dynamic calculation. At the beginning of each record, there are following three data.

```
Steps:int   // Time step in the dynamic calculation.
            // In the static calculation, it is set to be 0.
Time:double // Real time in the dynamic calculation.
            // In the static calculation, it is set to be 0.
Nscf:int    // The number of iterations performed until the SCF converged.
```

In all calculations, the initial state is first written with the record number 0, the step of dynamics 0, time 0, and the number of iterations for SCF 0. In the static equilibrium calculations, the final state after the convergence of the SCF is written with the record number 1, the step of dynamics 0, and time 0. In the dynamic calculation, the record at each time step is appended at the end of the file as the time step proceeds. The following describes the detail of the main part of the data. Readers should refer to Section 5.10 for the detailed definitions of the output UDF file.

```
SUSHIOutput: {
  // Volume fractions in the final state of the simulation ####################
    volume_fractions:VolumeFractions // This parameter is the same as that defined in
                                     // the common UDF output.
  // phi and V ########################################################################
    flag_of_dynamics:int         // The flag that specifies the type of the calculation.
        // 0 : Static equilibrium calculation.
        // 1 : Dynamic calculation.
    phi:ScalarField              // Array of the scalar field phi for each species.
    V  :ScalarField              // Array of the scalar field V for each species.
  // Free energy   ####################################################################
    free_energy:double           // Total free energy of the system.
```

```
                                // Refer to Section 2.4.
    excess_free_energy:double   // Excess free energy. This parameter is valid only
                                // for the grand canonical ensemble system with an
                                // interface.
  // Optional output ###########################################################
    optional_output:OptionalOutput
  // End situation #############################################################
    flag_of_convergence:int
        // 0 : initial condition, The result is not a calculation result.
        // 1 : normal calculation.
        // 2 : exceeded max number of SCF iteration.
        // 3 : failed to solve Poisson equation.
        // 4 : terminated by KILL file.
        // 5 : terminated by the message of "STOP" from message file.
        // 6 : terminated by the message of "END"  from message file.
        // 7 : terminated by unknown error.
}
```

5.10 Detailed definitions of the output UDF

5.10.1 Scalar fields

SUSHIOutput.phi
SUSHIOutput.V
SUSHIOutput.optional_output.segment_volume_fraction
SUSHIOutput.optional_output.electric_potential

The self-consistent field V, the segment volume fraction ϕ, and the average segment volume fraction defined in the input data are written in the following format.

```
class ScalarArray:{ // /
    comp[]:double         // Array of the data for each component on a mesh point.
}
class ScalarField:{ // /
    name:KEY              // Name :This parameter can be used when one searches for
                          // this element using Python scripts.
                          // Refer to the UDF manual for details.
    num_of_component:int  // Total number of species.
    value[]:ScalarArray   // Value of the ScalarArray.
};
```

5.10.2 Optional output

SUSHIOutput.optional_output

The output data which are optionally inputted to calculate are summarized in the data structure.

```
class OptionalOutput: {
    segment_volume_fraction:ScalarField
        // Volume fraction of segment species specified in
        // the SUSHIInput.properties.segment_volume_fraction_conditions[].
    radius_of_gyration[]:RadiusOfGyration
        // Radii of gyration.  Refer to the next section.
    monte_carlo_condition_of_polymer[]:MonteCarloConditionOfPolymer
```

```
        // The positions of junctions specified in
        // the SUSHIInput.external_conditions.monte_carlo_conditions.
    electric_potential:ScalarField
        // Electric potential is output when the
        // SUSHIInput.external_conditions.electric_conditions is specified.
}
```

Radii of gyration

SUSHIOutput.optional_output.radius_of_gyration[]
If the calculation of the gyration radius is specified in the input UDF file, the following data are written into the output UDF file.

```
RadiusOfGyration: {
    dim:int          // The spatial dimension of the system used in the calculation.
                     // This is equal to the dimension of the mesh.
    rg:double        // The value of the radius of gyration.
    rg_xyz[]:double  // The value of the radius of gyration in x, y, and z directions,
                     // respectively.
                     // The same numbers of data as the spatial dimension are written.
}
```

5.11 Log files

Two logs are created by SUSHI. One is the status of the calculations, which is an output to the standard output of the system. The contents of this log differ depending on whether the calculation is a static equilibrium or a dynamic calculation. The output file can be specified by the "-C" option in the command line of SUSHI. The other log is the status of convergence of the SCF iterations. This log is overwritten at every time step of the dynamic calculation. The default name of this log file is "input_file_name.usl". The contents of these two log files are explained below. Refer to Section 5.3 for the details of the options.

5.11.1 Standard output log

In both static equilibrium calculations and dynamic calculation, the version of SUSHI and the name of the engine are first written as the following manner.

```
SCFEngine Version 4.0 Revision 060620L, Octa Project
file names
input ab_ring.sin
output ab_ring_uot.udf
restart ab_ring_uar.udf
archives ab_ring_uar.udf
log ab_ring.usl
cout ab_ring.ual
stop ab_ring.stp
```

The meanings of the files listed above are as follows.

```
input     Input file
output    Output file
restart   Restart file
```

```
archives  Archives file
log       Log file of SCF iterations in the dynamic calculation
cout      Standard output file
stop      Emergency stop file
```

These files are the input file and the several default files used by SUSHI. Some of these files are not actually used. The restart file is not used if the restart is not specified. In case of a static equilibrium calculation, the log file of the SCF iterations for the dynamic calculation will not be generated.

Standard output log for static equilibrium calculation

The following data are output.

```
.....
nSCF 1005 nW 0 err  3.25043835e-05  nV 1 err  1.01382781e-04 nTotPhi 0 err  8.44784069e-05
nSCF 1006 nW 0 err  3.89468651e-05  nV 0 err  9.99588160e-05 nTotPhi 0 err  7.55554209e-05
nSCF    1006 FreeEnergy 1.49953121e-01   ExcessFE -2.50396581e-01
//!!!!!!!! SCF Convergence succeeded. : nSCF = 1006 ////////
cpu time 10 [sec]
```

The format of the data written at every SCF iteration step is as follows.

```
    nSCF [Number of iterations in the current SCF calculation]
    nW   [Number of mesh points at which W is not converged] err [Maximum error in W]
    nV   [Number of mesh points at which V is not converged] err [Maximum error in V]
    nPhi [Number of mesh points at which the incompressible condition is not satisfied]
         err [Maximum error in the sum of the volume fractions]
```

If the maximum error becomes smaller than the threshold value specified in the input UDF data, the iteration will be terminated. Readers should refer to Section 2.9.9 for the details. The number of iteration steps for the SCF, the free energy, and the excess free energy are output at the end. In case that the SCF iterations are normally terminated, the following message is written.

```
//!!!!!!!! SCF Convergence succeeded. : nSCF = 1006 ////////
```

On the other hand, in case that the SCF iterations are abnormally terminated, the following messgae is written.

```
//XXXXXXXX Exceeded maximum step number of SCF. : nSCF = 20000 ////////
```

The last line is the actual elapsed time.

Standard output log for the dynamic calculation

The following data are output.

```
.....
nStep 499999 time 499.999 nSCF 2 F -2.14330212e-01  ExcessFE -1.60555993e+01
nStep 500000 time 500     nSCF 3 F -2.14330226e-01  ExcessFE -1.60556145e+01
//!!!!!!!! SCF Convergence succeeded. : nSCF = 3 ////////
cpu time 20614 [sec]
```

The format of the data written at every time step of the dynamic calculation is as follows

```
    nStep [Time step of the dynamic calculation] time  [Time]
    nSCF  [Number of iterations of the SCF calculation]
    F     [Free energy / volume]  ExcessFE [Excess free energy]
```

If the SCF iteration did not converge, the following message will be written, and SUSHI is terminated.

```
//XXXXXXXX Exceeded maximum step number of SCF. : nSCF = 20000 ////////
```

The last line is the actual elapsed time.

Log file of SCF iterations in dynamic calculation

The contents of this file are as follows.

```
nSCF 1 not equilibrated Polymers 2
    polymerId   0  nW       0  err 0.00000000e+00    nPhi       4  err 1.42054109e-04
    polymerId   1  nW       0  err 0.00000000e+00    nPhi       7  err 1.38242704e-04
.....
```

The first line is the number of polymer species that have not yet converged, followed by the messages with the format below repeatedly.

```
    polymerId [ID of the polymer]
    nW   [Number of mesh points at which W is not converged] err [Maximum error in W]
    nPhi [Number of mesh points at which phi is not converged]  err [Maximum error in phi]
```

When the maximum error becomes smaller than a threshold value specified in the input file, the calculation will be terminated. Refer to Section 2.9.10 for the details.

5.12 Parameter UDF

When the user starts SUSHI with the option "-Pparameter_udf_file", the UDF control parameters stored in the file "parameter_udf_files" are read. The definitions of the UDF control parameters are as follows.

```
\begin{header}
\begin{def}
EngineType:string;
EngineVersion:string;
IOType:string;
ProjectName:string;
Comment:string;
\end{def}
\begin{data}
EngineType:"SUSHI"
EngineVersion:"060620"
IOType:"IN"
ProjectName:"WG2"
Comment:"Control Parameter"
```

```
\end{data}
\end{header}
\begin{def}
Time:double
class ChiParameter:{
    name_i:string
    name_j:string
    parameter:double
}
///// START SUSHIParameter.udf ////////////////////////////////////////////////
SCFControlParameter:{
    constV                          :double
    constW                          :double
    error                           :double
    judge_method                    :select { "ABSOLUTE", "RELATIVE" }
    scf_output_interval_step        :int
    max_SCF_step                    :int
}
DynamicsControlParameter:{
    delta_t                         :double
    max_dynamics_step               :int
    output_interval_step            :int
    archives_interval_step          :int
    log_interval_step               :int
}
VariableControlParameter:{
    chi_parameters[]:ChiParameter
}
///// END SUSHIParameter.udf //////////////////////////////////////////////////
\end{def}
```

The meanings of the parameters are the same as those in the SUSHIInput.udf file. To control SUSHI by the parameters, you have to prepare the file named "parameter_udf_file" in which the values of the parameters that should be changed are stored in the above UDF format. SUSHI reads this file and change the values of the parametrers when the specified time in this parameter UDF file matches the simulation time. In the case of a static equilibrium calculation, only time = 0 can ce specified in the parameter UDF file. In the case of a dynamic calculation, you can prepare many records changing the parameters and the time. For example, a simulation of the quenching process can be simulated by changing the value of χ parameter. This can be realized by using the parameter UDF file.

Alphabetical UDF classes	
UDF parameter	Description
ADFParameter	class ADFParameter
alpha	double
allowedError	double
maxIteration	int
alphaForPoissonSolver	double
allowedErrorForPoissonSolver	double
maxIterationForPoissonSolver	int
bondFactor	double
bulkModulus	double
phiMin	double
phiMax	double
isIncompressible	int (no effect)
AxisBoundaryCondition	class AxisBoundaryCondition
axis_conditions[]	array of string PERIODIC/DIRICHLET/WALL/NEUMANN
AxisRegion	class AxisRegion
axis_name	[select] X/Y/Z/R/H
Block	class Block
monomer_name	string
number_of_monomer	double
BoundaryCondition	class BoundaryCondition
conditions[]	array of AxisBoundaryCondition class
volume_fractions_on_boundaries	VolumeFractionsOnBoundaries class
CalculationMethod	class CalculationMethod
type	[select] STATICS/DYNAMICS/MONTECARLO
ChiParameter	class ChiParameter
name_i	string
name_j	string
parameter	double
Components	class Components
polymers[]	array of Polymer class
solvents[]	array of Solvent class
ConstraintCondition	class ConstraintCondition
target_ID	SCFUnitIDSet class
constraint_coondition	int 1: phi/ 2: V
DomainSpecificationCondition	class DomainSpecificationCondition
name	string
domain_regions[]	array of AxisRegion class
initial_chemical_potential	double

UDF parameter	Description
DynamicConditions	class DynamicConditions
segment_mobilities[]	array of SegmentMobility class
types_of_polymer_mobility[]	array of LocalMobility class
types_of_solvent_mobility[]	array of LocalMobility class
reaction_conditions_of_rapid_reactions[]	array of ReactionConditionOfRapidReaction class
reaction_conditions_of_active_sites[]	array of ReactionConditionOfActiveSites class
reaction_conditions_of_grafts[]	array of ReactionConditionOfGraft class
shear	class Shear
noise	class Noise
system_size_dynamics	class SystemSizeDynamics
DynamicsParameter	class DynamicsParameter
delta_t	double
max_dynamics_step	int
output_interval_step	int
archives_interval_step	int
log_interval_step	int
dynamics_scheme	[select] EXPLICIT/EXPLICIT2/IMPLICIT
compressibility	double
Electrolyte	class Electrolyte
name	string
name_of_dissociated_ion	string
name_of_free_ion	string
kp	double
ElectrostaticCondition	class ElectrostaticCondition
dielectric_constant_of_system	double
electrostatic_parameters_of_segment[]	array of ElectrostaticParametersOfSegment class
electrolyte[]	array of Electrolyte class
external_electric_field	Vector3D class
parameter_of_external_electric_field_for_dynamics	ParameterOfExternalElectricFieldForDynamics class
ElectrostaticConditions	class ElectrostaticConditions
electrostatic_condition	class ElectrostaticCondition
poisson_solver_parameter	PoissonSolverParameter class
ElectrostaticParametersOfSegment	class ElectrostaticParametersOfSegment
name	string
charge	double
dielectric_constant	double
ExternalConditions	class ExternalConditions
surface_chi_parameters[]	array of SurfaceChiParameter class
graft_conditions[]	array of GraftCondition class
mask_conditions[]	array of MaskCondition class
static_conditions	StaticConditions class
dynamic_conditions	DynamicConditions class
monte_carlo_conditions	MonteCarloConditions class
FHParameter	class FHParameter
kappas[]	array of Kappa class

UDF parameter	Description
Fiber	class Fiber
position_of_end0	Vector3D class
position_of_end1	Vector3D class
radius_of_fiber	double
end_cap	[select] YES/NO
region_of_obstacle	[select] IN/OUT
GraftCondition	class GraftCondition
polymer_ID	int
junction_ID	int
boundary_name	[select] XMin/XMax/YMin/YMax/ZMin/ZMax RMin/RMax/HMin/HMax/Particle/Fiber
obstacle_ID	int
Hexahedron	class Hexahedron
position_of_origin0	Vector3D
direction_a0	Vector3D
direction_b0	Vector3D
direction_c0	Vector3D
position_of_origin1	Vector3D
direction_a1	Vector3D
direction_b1	Vector3D
direction_c1	Vector3D
coefs_of_surface_chi[]	double
region_of_obstacle	[select] IN/OUT
HydrodynamicsParameters	HydrodynamicsParameters
density	double
viscosities[]	Viscosity
neglect_convection_term	int
dynamics_scheme	{select} EXPLICIT/EXPLICIT2/IMPLICIT
poisson_solver_parameter	PoissonSolverParameter class
Hybrid	Hybrid
SCF_step	int
RPA_step	int
free_energy_by	{select} RPA/SCF
JunctionData	class JunctionData
ID_set[]	array of int
JunctionPair	class JunctionPair
first	int
second	int
Kappa	class Kappa
polymer_ID	int
value	double
LocalMobility	class LocalMobility
component_ID	int
type	[select] ROUSE/REPTATION

UDF parameter	Description
MaskCondition	class MaskCondition
polymer_ID	int
junction_ID	int
mask_regions[]	array of AxisRegion class
Mesh	class Mesh
name	KEY
type	[select] REGULAR/RECTANGULAR CYLINDRICAL/SPHERICAL
MeshAxis	class MeshAxis
values[]	array of double
Monomer	class Monomer
species_name	string
specific_volume	double
effective_bond_length	double
MonomerSCFChar	class MonomerSCFChar
name	string
states[]	array of State class
array_of_transition_state_probabilities[]	array of TransitionStateProbabilities class
MonteCarloConditionOfJunction	class MonteCarloConditionOfJunction
junction_ID	int
position	Vector3D class
MonteCarloConditionOfPolymer	class MonteCarloConditionOfPolymer
polymer_ID	int
monte_carlo_conditions_of_junction[]	array of MonteCarloConditionOfJunction class
MonteCarloConditions	class MonteCarloConditions
monte_carlo_conditions_of_polymer[]	array of MonteCarloConditionOfPolymer class
MonteCarloParameter	class MonteCarloParameter
max_monte_carlo_step	int
output_interval_step	int
archives_interval_step	int
log_interval_step	int
Noise	class Noise
standard_deviation_of_thermal_noise	double
Obstacles	class Obstacles
ndiv_for_volume_calculation	int
particles[]	array of Particle class
fibers[]	array of Fiber class
hexahedrons[]	Hexahedron
shape_by_grids[]	ShapeByGrids
OptionalOutput	class OptionalOutput
segment_volume_fraction	ScalarField class
radius_of_gyration[]	array of RadiusOfGyration class
monte_carlo_condition_of_polymer[]	array of MonteCarloConditionOfPolymer class
electric_potential	ScalarField class

UDF parameter	Description
Particle	class Particle
position_of_center	Vector3D class
radius_of_particle	double
region_of_obstacle	[select] IN/OUT
Path	class Path
polymer_ID	int
subchain_ID	int
begin_junction_ID	int
end_junction_ID	int
coef_for_surface_depth	double
name_as_solvent	string
effective_diffusion_constant	double
ParameterOfExternalElectricFieldForDynamics	class ParameterOfExternalElectricFieldForDynamics
direction	[select] X/Y/Z
alpha	double
PoissonSolverParameter	class PoissonSolverParameter
omega	double
allowed_error	double
max_iteration	int
PolydispersityCondition	class PolydispersityCondition
longest_homo_polymer_ID	int
target_homo_polymer_ID	int
Polymer	class Polymer
type	[select] HOMO/BLOCK/COMB/STAR/GENERAL
Properties	class Properties
segment_volume_fraction_conditions[]	array of SegmentVolumeFractionCondition class
radius_of_gyration_conditions[]	array of SubchainUnit class

UDF parameter	Description
RadiusOfGyration	class RadiusOfGyration
dim	int
rg	double
rg_xyz[]	array of double
ReactantIDSet	class ReactantIDSet
type	int 0 : polymer/ 1 : solvent
IDSet[]	array of int: molecular ID, junction ID
ReactionConditionOfActiveSites	class ReactionConditionOfActiveSites
reactant_IDs[]	array of ReactantIDSet class
polymer_ID_of_product	int
map_of_subchains[]	array of SubchainMap class
reaction_constant	double
ReactionConditionOfGraft	class ReactionConditionOfGraft
graft_condition	GraftCondition class
polymer_ID_of_product	int
map_of_subchain_IDs[]	array of int
reaction_constant	double
ReactionConditionOfRapidReaction	class ReactionConditionOfRapidReaction
reactant_IDs[]	array of ReactantIDSet class
product_ID	ReactantIDSet class
volume_fractions_of_reactants_in_product[]	array of double
reaction_constant	double
RPAParameter	RPAParameter
random_seed	int
standard_deviation	double

UDF parameter	Description
ScatteringFunctionInput	ScatteringFunctionInput
calculate	[select] ON/OFF/RESTAR
intensity	[select] ON/OFF
coef_for_mesh	double
SCFParameter	class SCFParameter
delta_s	double
constV	double
constW	double
error	double
random_seed	int
standard_deviation	double
method_of_convergence_test	[select] ABSOLUTE/RELATIVE
convergence_test_interval_step	int
max_SCF_step	int
scf_output_interval_step	int
SCF_method	[select] INCORE/DIRECT
pathintegral_scheme	[select] EXPLICIT/IMPLICIT
SCFUnitData	class SCFUnitData
name	string
volume_fraction	double
ID_set[]	array of int
SCFUnitIDSet	class SCFUnitIDSet
type	int 0 : polymer/1 : solvent
IDSet[]	array of int: molecular ID, subchain ID, state ID
ShapeByGrids	class ShapeByGrids
input_filepath	string
region_of_obstacle:select	[select] IN/OUT
SUSHIInput	class SUSHIInput
calculation_method	CalculationMethod class
restart	[select] START/CONTINUE/RESTART/RESTART_READMESH
solver_parameter	SolverParameter class
mesh	Mesh class
type_of_free_propagator_of_regular_mesh	[select] 1NN-P/2NN-NP/2NN-P/3NN-P
SUSHIOutput	class SUSHIOutput
volume_fractions	VolumeFractions class
flag_of_dynamics	int
phi	ScalarField class
V	ScalarField class
free_energy	double
excess_free_energy	double
optional_output	OptionalOutput class
flag_of_convergence	int
ScalarArray	class ScalarArray
comp[]	array of double
ScalarField	class ScalarField
name	KEY
num_of_component	int
value[]	array of ScalarArray class
SegmentMobility	class SegmentMobility
segment_name	string
mobility	double

UDF parameter	Description
SegmentVolumeFractionCondition	class SegmentVolumeFractionCondition
polymer_ID	int
subchain_ID	int
begin_length	double
end_length	double
Shear	class Shear
shear_rate	double
shear_period	double
Solvent	class Solvent
name	string
specific_volume	double
SolverParameter	class SolverParameter
type	[select] ADF/FH/SCF
State	class State
name	string
probability	double
StaticConditions	class StaticConditions
polydispersity_conditions[]	array of PolydispersityCondition class
symmetry_conditions[]	array of SymmetryCondition class
domain_specification_conditions[]	array of DomainSpecificationCondition class
constraint_conditions[]	array of ConstraintCondition class
SubchainMap	class SubchainMap
reactant_ID	int
subchain_ID_of_reactant	int
subchain_ID_of_product	int
SubchainUnit	class SubchainUnit
polymer_ID	int
subchain_ID	int
SurfaceChiParameter	class SurfaceChiParameter
boundary_name	[select] XMin/XMax/YMin/YMax/ZMin/ZMax RMin/RMax/HMin/HMax
obstacle_ID	int
SymmetryCondition	class SymmetryCondition
parent_path	Path class
child_path	Path class
SystemSizeDynamics	class SystemSizeDynamics
parameter_of_system_size_dynamics	double
compressibility	double
interval_of_system_size_dynamics	int
SystemSizeOptimaization	class SystemSizeOptimaization
max_iteration_of_optimization	int
cubic_system	int 0:no / 1:yes
TransitionStateProbabilities	class TransitionStateProbabilities
probabilities[]	array of double

UDF parameter	Description
Vector3D	class Vector3D
x	double
y	double
z	double
VolumeFraction	class VolumeFraction
Id	int
volume_fraction	double
ensemble	[select] CANONICAL/GRANDCANONICAL
VolumeFractionData	class VolumeFractionData
name	string
volume_fraction	double
VolumeFractionOnBoundary	class VolumeFractionOnBoundary
Id	int
boundary_name	[select] XMin/XMax/YMin/YMax/ZMin/ZMax RMin/RMax/HMin/HMax
VolumeFractions	class VolumeFractions
polymer_volume_fractions[]	array of VolumeFraction class
solvent_volume_fractions[]	array of VolumeFraction class
VolumeFractionsOnBoundaries	class VolumeFractionsOnBoundaries
polymer_volume_fractions[]	array of VolumeFractionOnBoundary class
solvent_volume_fractions[]	array of VolumeFractionOnBoundary class

5.13 Definitions of the SEED format

The SEED(Simple and Easy Editable Data) format is the format for simple data input. This format has been prepared for the development of the core of SUSHI, and is expected to be still useful for the future developments. In order to handle the SEED format files, the multimap and the string of STL of C++ language are extensively used. The SEED format is not suitable for a large-scale input data because of the extensive requirements for the cpu time and the memory. However, SEED format is useful for inputting small-scale data. The input SEED file is a UNIX text file. The text files with the MS-DOS format in which a line terminates with CR + LF cannot be accepted. A fundamental data sequence of the SEED file is as follows.

```
Key data1 data2 data3 ........
```

The first item "Key" is the keyword that is used to identify the data. "data1 data2 ..." are the data written in ASCII characters. The separator is either a blank character or a tab character. A single data sequence terminates with a "new-line" code. When a data includes two or more lines, the data sequence following Key have to be enclosed with "(","). " For example,

```
Key ( data1 data2 data3 .......
      data11 data12 data13 ........... )
```

Separators must be inserted after "(" and before ")", respectively. A data sequence is stored in a multi-map with the Key. Two or more data sequences can have the same Key. A data element in a data sequence is specified by the element index. Let us name the multi-map that contains the pair of Key and the data sequence as "mmapWords". The basic structure of the SEED format is a structure object that posesses this "mmapWords". Furthermore, the multi-map "mmapSeeds" that consists of Key and SEED is added to this structure object. Such a structure can easily be realized using the classes of C++ as follows.

```
class Seed {
    Seed* pParent;
    multimap< string, map< int, string > > mmapWords;
    multimap< string, Seed* >              mmapSeeds;
};
```

"pParent" is a pointer to the parent SEED. This pointer makes it possible to read any hierarchical data structures.

The data structure should be enclosed by "{", "}" as follows.

```
Key1 data1 data2 data3 ........
Key2 data1 data2 data3 ........
.....
Key3 {
    Key1 data1 data2 data3 ........
    Key2 data1 data2 data3 ........
    .......
    Key3 {
        ..........
    }
    ........
}
Key4 {
    ........
.......
```

As the data sequences and the SEED sequences are stored with Keys and multi-maps, the order of the data does not have a meaning except for the data with the same Key. Thus, a new data sequence or a new data structure with a different Key can be inserted in any position. For example,

```
Key1 data1 data2 data3
Key1 data4 data5 data6
Key2 data7 data8 data9
```

and

```
Key1 data1 data2 data3
Key2 data7 data8 data9
Key1 data4 data5 data6
```

are the same data. There is no fixed format in SEED. As long as there is no inconsistency in the structure, SUSHI can read the data and can extract required data.

5.14 Input method by SEED format

In the following, the symbols "....." mean repeating the same data or the same structure. The first word before the words enclosed by [] in each line is the key word, and the words enclosed by [] mean the data. This [meanings of data] part sould be replaced by a real value written in ASCII characters.

The input data for the basic controls are as follows.

```
SOLVER               [Solver]
    // The name of the solver, which means a special-purpose simulator/simulation method.
    // ADF : Approximate Density Functional method (Under construction).
    // SCF : Self Consistent Field method.
RPA_SCF   [Number of SCF iter.] [Number of RPA iter.]  // Hybrid method
CALCMETHOD           [The type of the calculation]
    // DYNAMICS   : Dynamic calculation.
    // STATICS    : Static equilibrium calculation.
    // MONTECARLO : Monte Carlo calculation.
                    // For selecting this option, one has to set parameters to
                    // MONTECARLO { ... } data.
RESTART
        // The flag for restarting condition
        // CONTINUE         : continue with reading the mesh at the final recoerd
        // no argument      : restart without reading the mesh at the final recoerd
        // RESTART          : restart without reading the mesh at the final recoerd
        // RESTART_READMESH : restart with reading the mesh at the final recoerd
```

5.14.1 Keys for the SCF calculation

```
DEL_S                [Mesh width along the chain contour used for the path integral
                      calculations]
CONST_V              [Constant for the updating of the chemical potential]
CONST_W              [Constant for the updating of the interaction]
ERROR                [Threshold used in the judgement of the convergence]
RANDOM_SEED          [Seed for random number]
STANDARD_DEVIATION   [Standard deviation of the Gaussian random noise for the
                      initial states of the fields]
JUDGEMETHOD          [Method of judging the convergence]
    // This parameter is valid only in the dynamic calculations.
    // ABSOLUTE   : The absolute error in phi is used in judging the convergence.
    // RELATIVE   : The relative error in phi is used in judging the convergence.
    //              This judgment is severer than the ABSOLUTE case.
JUDGESTEP             [Interval of judging the convergence]
MAX_COUNT             [Maximum iterations of the SCF calculation]
SCF_OUTPUTSTEP        [Output interval steps in the SCF calculation]
MATRIX_ELEMENT_TYPE   [Type of the discretized Laplacian]
    // 1NN-P, 2NN-NP, 2NN-P, 3NN-P.
SCFMETHOD             [Treatment of the memory for the path integral]
    // INCORE   : Data are stored on the memories.
    // DIRECT   : Data are re-calculated when they are required.
```

```
PATHINTEGRAL_SCHEME  [Integration scheme of the path integral]
    // IMPLICIT
    // EXPLICIT : default
```

5.14.2 Keys for the dynamic calculation

These Keys are valid in the dynamic calculations. Refer to Section 2.9.10.

```
DEL_T                [Time mesh width]
NTIME                [Maximum number of time steps]
OUTPUTSTEP           [Output interval]
ARCHIVESSTEP         [Output interval steps to the archives file]
LOGSTEP              [Output interval steps to the log file]
DYNAMICS_SCHEME      [Integration scheme of the equation of motion]
    // IMPLICIT
    // EXPLICIT // The explicit scheme. default. Euler scheme.
    // EXPLICIT2// The2step Runge-Kutta scheme.
 COMPRESS_DY          [Compressibility : Available in SCF calculation.]
```

5.14.3 Keys for the Monte Carlo calculation

These Keys are valid in the Monte Carlo calculations and in common with the keys of dynamics calculation. Refer to Section 2.9.17.

```
NTIME                [Maximum number of steps]
OUTPUTSTEP           [Output interval]
ARCHIVESSTEP         [Output interval steps to the archives file]
LOGSTEP              [Output interval steps to the log file]
```

5.14.4 Mesh

Refer to Section 2.9.3.

Mesh

```
MESH {
    NAME [Name]
    TYPE [Type]
         // REGULAR
         // RECTANGULAR
         // SPHERICAL
         // CYLINDRICAL
    X [Minimum value of X-axis] [Maximum value] [Number of cells] [Number of regions(MPI only)]
    Y [Minimum value of Y-axis] [Maximum value] [Number of cells] [Number of regions(MPI only)]
    Z [Minimum value of Z-axis] [Maximum value] [Number of cells] [Number of regions(MPI only)]
    X [Array of the positions of each mesh points] .....(rectangular mesh case)
    Y [Array of the positions of each mesh points] .....(rectangular mesh case)
    Z [Array of the positions of each mesh points] .....(rectangular mesh case)
    R [Minimum value of R-axis] [Maximum value] [Number of cells]
    H [Minimum value of H-axis] [Maximum value] [Number of cells]
}
```

Boundary conditions

Refer to Section 2.9.4.

```
BOUNDARYCONDITION {
    X [Condition at the minimum end of X-axis] [Condition at maximum end]
```

```
    Y [Condition at the minimum end of Y-axis] [Condition at maximum end]
    Z [Condition at the minimum end of Z-axis] [Condition at maximum end]
    R [Condition at the minimum end of R-axis] [Condition at maximum end]
    H [Condition at the minimum end of H-axis] [Condition at maximum end]
         // PERIODIC          :Periodic boundary conditions.
         //                    In this case, you do not have to specify
         //                    the conditions for both minimum and maximum ends.
         //                    Only a single datum is enough.
         // DIRICHLET or WALL :Absorbing wall.
         //                    Dirichlet boundary condition with constant 0.
         // NEUMANN           :Reflective wall.
         //                    Neumann boundary condition with constant gradient 0.
         // The conditions for both ends should be specified except for
         // the PERIODIC boundary case.
    VOLUME {  // The volume fractions at the boundary.
              // Refer to the UDF definition.
        POLYMER {
            FRACTION [ID of polymer] [End of axis] [Volume fraction]
            .....
                // ID is the element index defined in COMPONENT. It starts from 0.
                // The value of the volume fraction is constrained on the
                // plane that goes through the end point of the axis and is
                // perpendicular to the axis.
                // XMin, XMax, YMin, YMax, ZMin, ZMax, RMin, RMax, HMin, Hmax
        }
        SOLVENT {
            FRACTION [ID of solvent] [End of axis] [Volume fraction]
            .....
                // ID is the element index defined in COMPONENT. It starts from 0.
                // The value of the volume fraction is constrained on the
                // plane that goes through the end point of the axis and is
                // perpendicular to the axis.
                // XMin, XMax, YMin, YMax, ZMin, ZMax, RMin, RMax, HMin, Hmax
        }
    }
};
```

5.14.5 Specification of the monomers

Refer to Sections 2.2 and 2.3.

Monomer

```
MONOMER {
    CHARACTER [Name] [Specific volume] [Effective bond length]
    .....
        // Name:
        //     If the monomer has its internal states such as the tapered block
        //     copolymer, this parameter specifies the sequence of the monomers
        //     such as "taperAB".
        // Specific volume:
        //     This parameter is used when the path integral and the volume fraction of
        //     the monomer are calculated.  If the monomer has its internal states, this
        //     parameter is re-calculated using the specific volumes of individual state,
        //     and will be overwritten.
        // Effective bond length:
        //     This parameter is used for the calculation of the coefficient of the
```

```
        //     Laplacian term in the evolution equation for the path integrals.
}
```

Internal states of subchains

These parameters specify the characteristic properties of the internal states of subchains. If none of the subchains has the internal state, this parameter can be skipped.

```
MONOMERSCFCHARTABLE {
    MONOMERSCFCHAR {
        NAME [Name]
            // A monomer with this name has to be defined in advance.
            // For example, a monomer that is composed of a tapered block of A and
            // B monomers may be named "taperedAB".
        STATE { // Array of the elements of the transition state probability matrix
                // for the internal states.
            DATA [Name] [Probability of finding this monomer in the subchain]
        .....
        }
        TRANSITIONSTATEMATRIX {
            NMATRIX [Number of transition state probabilities]
            PROBABILITY [Transition state probability] .......
                // Array of the elements of the transition state probability matrix between
                // the internal states of the subchain. The transition state probability
                // Tij is defined as the probability of finding a monomer with state j
                // next to a monomer with state i separated by the distance equal to the
                // mesh width along the chain. In the current version of SUSHI, only the
                // polymers with fixed concentration distribution of segment kind along
                // the chain can be calculated. Therefore, the element Tij is determined
                // only by the probability of finding the internal state j at the
                // specified position. The position is specified by the distance from the
                // junction with samller ID. The value of the distance is corresponds to
                // the product of the index of this data structure and the mesh width
                // along the chain ( DEL_S ).
                // The total number of array elements should coincide with
                // "( subchain length / delta_s )+ 1".
        }
    }
    .....
}
```

5.14.6 Specification of the polymers and solvents

These parameters specify the structures of the molecules (polymers and solvents). Only those whose volume fractions are specified are actually used in the simulation. Refer to Sections 2.2 and 2.3.

Component

```
COMPONENT {
    // Polymer is specified by connected subchains.  A subchain is specified by
    // a sequence of monomers with a certain length.
    POLYMER {
        TYPE [Key word of connection of the subchains]
        // HOMO  :Linear homo polymer.
        // BLOCK :Linear multiblock copolymer made of a sequence of the subchains
        //        with the same order as they are listed in BLOCK data line as follows.
        // STAR  :Star block copolymer made of subchains which are connected at
        //        a single junction point.
```

```
        // COOMB :Comb-type block copolymer.
             // The order of the subchains stored in the array is as follows.
             // main chain - side chain - main chain - side chain -.....
             // For example, an array of the subchains A1, B1, A2, B2, and A3
             // indicates the following structure.
             //     A1--+--A2--+--A3
             //         B1     B2
        //  GENERAL : The way how the subchains are connected each other
             //       is specified by the ID's of the two junction points at the
             //       both ends of each subchain.
        BLOCK [Name of the monomer of the subchain] [Length of the subchain]
        .....
            // The monomer with this name must be defined in MONOMER.
        JUNCTION [ID of junction] [ID of junction]
        .....
             // This parameter is available only when the type of the polymer is GENERAL.
             // The values of the ID must start from 0.
             // For example, when the number of elements of Block is 1 and a pair of ID's
             // is [0, 0], it means that this polymer is a ring polymer.
    }
    .....
    Solvent [Name] [Specific volume]
    .....
}
```

5.14.7 Volume fractions

Among the polymers and solvents defined in COMPONENT, only those whose volume fractions are specified are used in the simulation. Depending on the assumed statistical ensembles, the meanings of the volume fractions change. Refer to Section 2.9.8.

```
VOLUME {
    POLYMER {
        FRACTION [ID of polymer] [Volume fraction] [Ensemble] [Volume fraction of bulk]
        .....
            // ID is the element index of this component in the array COMPONENT.
            // It starts from 0.
            // Ensemble:
            //     CANONICAL      :Canonical ensemble.
            //     GRANDCANONICAL :Grand canonical ensemble.
            // Dynamics calculation cannot be performed when GRANDCANONICAL is selected.
    }
    SOLVENT {
        FRACTION [ID of solvent] [Volume fraction] [Ensemble] [Volume fraction of bulk]
        .....
            // ID is the element index of this component in the array COMPONENT.
            // It starts from 0.
            // Ensemble:
            //     CANONICAL      :Canonical ensemble.
            //     GRANDCANONICAL :Grand canonical ensemble.
            // Dynamics calculation cannot be performed when GRANDCANONICAL is selected.
    }
}
```

For the details, readers should refer to the explanations on the corresponding UDF data.

5.14.8 Interaction parameters between segments (χ parameters)

Refer to Chapter 7.

```
CHI {
    ZBE [Name of segment i] [Name of segment j] [Value of Chi_ij]
        // Chi_ji is automatically set using the symmetric relation
        // Chi_ji = Chi_ij.
        // If the value is not defined, it is assumed to be 0.
        // The name of the segment means the name of MONOMER or SOLVENT.
}
```

5.14.9 External conditions

The user can obtain several physical quantities which are specified using the ID's of the polymers, solvents, subchains, and junctions, which correspond to their element indices in the arrays defined in the SUSHIInput.components. Here, as the rule in C language, the element index begins from 0.

Volume fraction of a part of a subchain

The volume fraction of a specific section on a subchain can be calculated. This parameter is valid only for the polymers without a loop.

```
SEGMENT {
    SET [ID of polymer] [ID of subchain] [The starting position on the subchain]
        [The ending position on the subchain]
    .....
         // The smaller one of the two JUNCTION ID's of the both ends of the
         // subchain is used as the reference point for the position.
}
```

Radii of gyration (R_g)

This parameter specifies the subchains whose radii of gyration are calculated and output.

```
RG {
    ID [ID of polymer] [ID of subchain]  //  if the ID of subchain = -1, calculate the Rg of full
    .....
}
```

Scattering function

```
SCATT [Option of restart]
      //  RESTART : calculate scattering functions of existing records

      //            without date of the existing records
COEFFORMESHWIDTH [The coefficient for mesh width of 1D scattering function]
      // The mesh witdh = this value times 2 pi
      //                  / (the longest length of edge).
}
```

Interaction parameters between a wall and the segments (χ_s)

Refer to Section 2.9.7.

```
SURFACECHI {
    // The value of the interaction parameter between the segment and the wall.
    ZBE [End of axis] [Target segment name] [Value of parameter]
    .....
        // End of axis: Definition of the surface with which the segments interact.
```

```
        //    XMin, XMax, YMin, YMax, ZMin, ZMax, RMin, RMax, HMin, Hmax, PARTICLE, FIBER
        // ID of each obstacle (available when choosing PARTICLE or FIBER)
}
```

Graft conditions

A graft chain can be realized by imposing a constraint on the initial value of the path integral at the free end(s). Refer to Section 2.9.6.

```
GRAFT {
    // The wall onto which the chain is grafted and the grafted polymer are specified.
    SET [End of axis] [Polymer ID to graft] [JUNCTION ID to graft]
    .....
        // End of axis: The plane to which the polymer is grafted is defined as a plane
        //              that goes through the end point of an axis and is parpendicular
        //              to the axis.
        //    XMin, XMax, YMin, YMax, ZMin, ZMax, RMin, RMax, HMin, Hmax, PARTICLE, FIBER
        // ID of each obstqacle (availabe when choosing PARTICLE or FIBER)
}
```

Mask conditions

This parameter specifies the region within which the position of a free end is confined. This function can be used in simulating micelles, etc. Refer to Section 2.9.14.

```
MASK {
    JUNCTION {
        ID    [ID of the polymer to be masked] [ID of the junction to be masked]
        AXIS  [Name of axis] [Lower bound of the region] [Upper bound of the region]
        .....
            // Name of the axis:   X, Y, Z and R, H
    }
    .....
}
```

5.14.10 Effective external conditions in static equilibrium calculation

Specification of the longest polymer for the static equilibrium calculation on polydisperse linear homopolymer systems

This parameter is valid only for the static equilibrium calculations on systems composed of polydisperse linear homopolymers composed of a monomer with a single internal state. Specifying this parameter will reduce the memory and computational time. Refer to Section 2.9.12.

```
POLYDISPERSITY {
    SET [ID of the longest linear homopolymer] [Polymer ID of common path integral]
    .....
}
```

Specification of the similar path integrals for the static equilibrium calculation

Refer to Section 2.9.12. This parameter specifies the subchains whose path integrals can be shared each other. This parameter is valid only for the static equilibrium calculations. The calculation time of the path integrals will be reduced using this function. For example, in the calculation of two diblock copolymer chains $A_{10}B_{10}$ and $A_{20}B_{20}$, the path integral starting from the free end of the A subchain of the longer chain $A_{20}B_{20}$ is the same as that of the shorter chain $A_{10}B_{10}$, and can be shared. On the other hand, since the initial values of the path integral starting from the junction points are different for $A_{10}B_{10}$ and $A_{20}B_{20}$, these cannot be shared.

```
SYMMETRY {
    PATH [ID1] [ID2] [J1] [J2] [ID3] [ID4] [J3] [J4]
    .....
        // The path of the subchain whose path integral is shared with the other subchains.
        // The path are specified as follows.
        // ID1: ID of the polymer.
        // ID2: ID of the subchain.
        // J1 : ID of the junction from which the calculation of the path integral starts.
        // J2 : ID of the junction at which the calculation of the path integral ends.
        // The path of the subchain whose path integral is replaced by that defined above.
        // The path are specified as follows.
        // ID3: ID of the polymer.
        // ID4: ID of the subchain.
        // J3 : ID of the junction from which the calculation of the path integral starts.
        // J4 : ID of the junction at which the calculation of the path integral ends.
}
```

Domain specification

In order to produce a desired domain morphology, an appropriate initial condition for the self consistent field should be prepared using this parameter. This parameter is valid for the static equilibrium calculations. Refer to Section 2.9.13.

```
DOMAIN {
    SPECIFICATION {
        NAME [Name of segment]
        REGION [Name of axis] [Minimum value of the region where a small value is set as
                the initial SCF value] [Maximum value of the region]
        .....
            // Name of axis:   X, Y, Z and R, H
    }
    .....
}
```

Constraint conditions in the SCF calculation

This parameter specifies the constraint conditions imposed on the system when the SCF iteration is performed.

```
CONSTRAINT {
    POLYMER {
        TARGET [Object to be constrained] [ID of polymer] [ID of subchain] [ID of state]
        .....
            // [Object to be constrained]: 1  :The volume fraction of phi only.
    }
    SOLVENT {
        TARGET [Object to be constrained] [ID of solvent]
        .....
    }
}
```

Statical system size optimization

```
MAX_LO          [Interval SCF step of optimization]
DL              [Relative delta of length of edge, default:1e-6]
LATTICEERROR    [Threshold used in the judgement of the convergence of the length of the edge, defa
DUMPPARAM       [Allowed relative variation of the lengh of the edge, default:0.4]
```

5.14.11 Effective external conditions in dynamimc calculation

Mobility for each segment species

The users can specify the values of the mobility for each kind of monomer or solvent separately. If this parameter is not specified, the mobility is assumed to be 1.0. This parameter corresponds to the constant L_0 in Eqs.(2.64) and (2.65). Refer to Section 2.9.18.

The mobility of a polymer in a dilute solution in general depends on the local concentration of the polymer and the species of the surrounding matrix. In the dilute limit, the mobility can be written as $L_0\phi$, where the parameter L_0 depends on the properties of the chain and that of the matrix solvent. Refer to Section 2.9.10.

ROUSE case: When the chain length is much longer than that of the matrix solvent and the polymer concentration is small enough (dilute limit), the chain is in the Rouse regime. In such a case, the mobility L is given by $L = L_0\phi$.

REPTATION case: When the chain length is long and is almost equal to that of the matrix molecules, the chains are strongly entangled. In such a case, the mobility is given by $L = (L_0/N)\phi$, where N is the length of the target chain. This REPTATION case is not valid for a solvent.

```
MOBILITY {
    SEGMENT [Name] [Mobility]         // The value of L0.
    POLYMER [ID of polymer] [Type]    // The type of the mobility for the polymer.
      // ROUSE /
      // REPTATION/
    SOLVENT [ID of solvent] [Type]    // The type of the mobility for the solvent.
      // ROUSE -- only this condition is valid.
}
```

Chemical reactions

A chemical reaction can be introduced into the dynamic calculations. Available types of chemical reactions are follows.

1) $A + B + C + \cdots \rightarrow D$, where each species has to be defined in COMPONENT.

2) Reactions at active sites. For example, free ends of an A-homopolymer and a B-homopolymer react to produce an AB block copolymer.

3) Grafting reaction onto a wall.

1) $A + B + C + \cdots \rightarrow D$

```
If the characteristic time scale of the chemical reaction is faster than the time mesh
size used in the dynamic calculation, it is reasonable to assume that a certain amount
of the reactants turns into the same amount of the product instantaneously.
```

```
REACTION {
    GROUP {
        REACTANT [type of reactant] [ID of reactant]
        .....
        PRODUCT  [type of product] [ID of product]
                 [Volume fraction of the reactant listed above REACTANT line
                  in this product] .....
        CONSTANT [Reaction rate constant]
    }
    .....
        // Type of reactants :POLYMER, SOLVENT
}
```

2) Reactions at active sites

```
This is the case of the reactions with structural change of the molecules.
In this case, the product must be a polymer. Numbers of reactant species are limited to two.
Therefore, the reaction should be a second order one.

REACTION {
    JUNCTION {
        REACTANT [Type of reactant] [ID of reactant] [ID of junction of product]
        REACTANT [Type of reactant] [ID of reactant] [ID of junction of product]
          // Type of reactant :POLYMER, SOLVENT
        PRODUCT  [ID of product]
          // This parameter specifies the correspondence between the subchain in the
          // reactant and that in the product.
        SUBCHAIN [ID of reactant] [ID of subchain in the reactant]
                 [ID of subchain in the product]
          // ID of the reactant means the element index of the REACTANT specified
          // in the first line. This parameter can be either 0 or 1.
        .....
        CONSTANT [Reaction rate constant]
    }
    .....
}
```

3) Grafting reactions onto a wall

```
REACTION {
    GRAFT {
        REACTANT [ID of reactant] [ID of product]
        PRODUCT  [ID of product]
        SUBCHAIN [ID of the subchain in the product] ......
          // A list of ID's of subchains in the reactant corresponding to
          // the list of ID's of subchains in the product.
        CONSTANT [Reaction rate constant]
    }
    .....
}
```

Sheared dynamics

```
SHEAR_RATE  [Shear rate] [Period]
            // When the period is zero, one direction shear is added.
```

Thermal fluctuation

```
NOISE  [Standard deviation of thermal noise] [random_seed:0 or neglect is reccomended]
random_seed // If a positive value is set, the value is fixed and is not changed
                    // thus the positive value is used when only debugging.
```

Dynamic system size optimization

Refer to Section 2.9.18.

```
LATTICEOPT_COEF  [Q in eq. (2.103)]
COMPRESS         [Compressibility in eq. (2.104), default 0.]
LO_INTERVALSTEP  [Interval of dynamic step of optimization]
SSO {
    DIRECTION [flag of X axis] [flag of Y axis] [flag of Z axis]
    ....
```

```
    // If you optimize x and y axes keeping the ratio, input as
    // DIRECTION 1 1 0
    // If you optimize x and y axes independently, input as
    // DIRECTION 1 0 0
    // DIRECTION 0 1 0
}
```

Hydrodynamic effects

Introduce hydrodynamic effects for dynamics calculations, refer to Section 2.8.5.

```
HYDRO {
    DENSITY [Denisty]
    VISCOSITY [Name of segment]   [Viscosity]
    ....
    IMPLICIT  //Write this keyword when IMPLICT is used.
    POISSON { // Parameter for solve pressure
        ERROR     [Allowed error; about 10^-10 is recommended]
        MAXSTEP   [Maxinum number for ICCG]
    }
    // The parameter set to add a thermal fluctuation to the Navier-Stokes equation.
    NOISE  [Standard deviation of thermal noise] [random_seed:0 or neglect is reccomended]
}
```

5.14.12 Conditions for SCF Monte Carlo calculation

Refer to Section 2.9.17. Set the parameter of CALCMETHOD as "MONTECARLO" to use this option.

```
MONTECARLO {
    POLYMER { // Specify the polymer whose junctions are updated
              // in the Monte Carlo simulation.
        ID [number of ID of polymer]
        JUNCTION [number of ID's of junctions] [cordinate x] [cordinate y] [cordinate z]
        .....
             // Set the initial positions of junctions.
    }
    .....
}
```

5.14.13 Electrostatic condition

Refer to Section 5.7.11.

```
ELECTROSTATIC {
    DIELECTRIC [the value of the dielectric constant of the system]
        // It must be positive.
    CHARGE [name of segment] [value of charge] [dielectric constant of segment]
    .....
    ALPHA [direction] [value of alpha]
        // The paramers of extenal electric field for Dynamics.
        // direction:   X, Y, and Z.
    EFIELD [x value of E0]  [y value of E0]  [z value of E0]
        // The vector of extenal electric field.
        // Attention! ALPHA and EFIELD can not be used at the same time.
    POISSON { // Parameters for Poisson solver of SOR( Successive Over Relation) method.
        OMEGA [omega] // The parameter for SOR method.
                      // The value 0.8 is recommended as the default value.
                      // This paramerer is not used in current version SUSHI.
```

```
        ERROR  [value of error for the judgement of convergence ]
                      // The value 10^-10 (0.0001 written in previous manuals is bug)
                      // is recommended as the default value.
        MAXSTEP [the maximum number of iteration]
                      // The value 10000 is recommended as the default value.
    }
}
```

5.14.14 Obstacles

```
OBSTACLE {
    NDIV [Number of division of per mesh to get the boundary of obstacles.]
         // About 1000 is recommended.
    PARTICLE [X] [Y] [Z]  [radius_of_particle]  [IN or OUT]
         // Center  position of the particle
         // In or Out:Inner region or Outer region of the shell of the particle.
    FIBER [X] [Y] [Z] // coordinate one end
          [X] [Y] [Z] // coordinate another edn
          [radius_of_fiber]
          [IN or OUT]                          [IN or OUT]
          // In or Out:Inner region or Outer   // end cap
          // region of the shell of the fiber.
    .....
}
```

5.15 Limitations of parallel computation

There are limitiatiosn of parallel computation in SUSHI.

1. Deleted:Only periodic boundary condition is available.
2. Modify:It is recommended to use multiples of 8 for the number of division (number of cell) of all axes in GPU calculation.
3. Deleted:The number of division of all axes for MPI must be larger thean 1 to make regions.
4. SSO is not available.
5. Setting of obstacles is not available.
6. Deleted:GRPA dynamics can not be used with MPI.

When SUSHI encounters to unabel scheme. SUSHI stops with warning. These limitations will be removed in future.

Chapter 6

Useful Python Scripts

Almost *.py scripts were moved to action files(*.act). Following descriptions of *.py are old references. Please refer "action" section to do run *.py scripts.

6.1 What is showed?

In SUSHI, the minimum unit for which the volume fractions can be specified is the SCF unit. You can observe the density profile of the specified SCF unit, or the profile of a combination of several SCF units. Please refer to Section 5.8.2 for the definition of the SCF unit and the data structure "Composition", which stores the results of the analyses on the compositions of each SCF unit. The SUSHIOutput.phi is the data structure including the volume fraction fields of the SCF units.

6.2 sushi_show.py

The Python script named "sushi_show.py" is a tool for the visualization of the volume fraction fields obtained with the SUSHI. As is shown in the following, the behavior of "sushi_show.py" changes depending on the dimensionality of the simulation system. This script is valid for the regular mesh, cylindrical coordinate mesh, and the spherical coordinate mesh. At first, you need to jump to the record you want to display the result.

6.2.1 One-dimensional case

In this one-dimensional case, when you click the "Run" button, a graph of the profile of the volume fraction ϕ is obtained using the software "Gnuplot". A set of messages will appear in the bottom window. These messages are commands for Gnouplot.

Selection of the SCF unit

Set the ID of the SCF unit to the "component_list" in the python script. For example, when you want to show the density profile of phi fields with ID 0 and 1, set the data as
component_list = [[0], [1]].
The inner "[]" is needed to combine the plural phi fields. When you want to show the sum of the SCF units with the ID's 0 and 1 and also the SCF unit with ID 2, set the data as
component_list = [[0,1], [2]].
Any way, do not forget the inner "[]". The default is
component_list = []
which means that all phi fields are drawn.

Setting the value of axes

Set the values to xmin, xmax, ymin, and ymax to specify the range to be plotted. The default values for the X axis are

xmin = 0.
xmax = 0.
which means that the whole range along the axis X from its minimum value to its maximum value is specified.

The plotted area can be changed also using the Gnuplot commands generated by the above procedure. The procedure is a little bit complicate, please refer as followings if you interest in. In the bottom window, a series of commands are shown. These commands are the Gnuplot commands that draws the graph you are looking at. Copy whole messages to memory buffer (If you use Windows system, select the messages and push ctrl-C). Next, click the "Plot" tab and move to the Plot window. Click the GraphSeet[] icon in the upper left window to make it effective. Click the "Make" button. This operation sets up the data file to be plotted and several messages will be displayed in the "Plot" window as default plotting commands. As you have already copied special plotting commands to the memory buffer, you can use them instead of these default ones. Then, delete these output lines by using "Clean" button and paste the special plotting commands stored in the memory buffer to this "Plot" window (If you use Windows system, push ctrl-V). After change the commands, clicking the "Plot" button, you will obtain a graph you want. Please refer to the Gnuplot manual about the commands.

6.2.2 Two- and three-dimensional cases

After loading the "sushi_show.py" script, click "Run" button on the "Window/Viewer" window. Then, the volume fraction profile will be drawn. In the default action, $\phi[0]$ (the volume fraction of the first SCF unit) is drawn. If you want to show the volume fraction of another SCF unit, you have to modify the python script. You can find the following line.
componentCoef = [0]
Replace the argument "0" with the ID of the species that you want to show. If you specify a list of ID's instead of a single ID, the sum of the volume fractions for the listed species will be shown. And the negative value of ID is valid, which add the negative value of the field. For example,
componentCoef = [0, -1]
means that the profile of phi[0] - phi[1] is shown.

The regions with larger volume fraction and with smaller volume fraction are shown in blue and red, respectively. The range to be shown is set automatically between the minimum value and the maximum value of the data. In order to change this range, delete the "#" to validate the message
`#crange=( 0., 1. )`
and change the two values in the "()", which specify the minimum value and the maximum value, respectively.

6.3 sushi_show3color.py

This script is used to show the domain structures of three component systems. A color is attributed to each of the three components. Each cell of the simulation box is painted with the color corresponding to the component that has the highest concentration among the three in the cell. This script is valid for both two and three dimensional systems with the regular mesh. The usage is the same as that for "sushi_show.py", i.e., simply clicking "RUN" button.

6.4 sushi_show_surf.py

The interfacial distributions in three-dimensional systems can be shown. The usage is the same as that for "sushi_show.py", i.e., simply clicking "RUN" button.

6.5 action

When the cursor is on SUSHIOutput icon as shown in the next figure, you can call an action python script file by clicking the right button of your mouse. In order to show the results, this method is more useful than using the python script files mentioned in the previous sections.

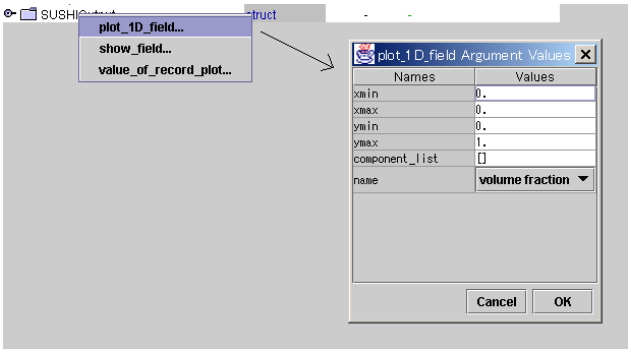


Figure 6.1: Select box of the action file

6.5.1 plot_1D_field

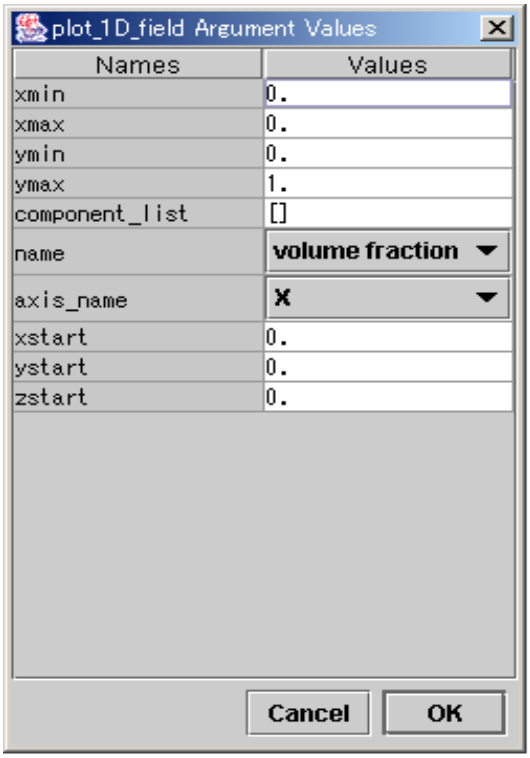


Figure 6.2: Parameters in the action box for "plot 1D field"

1) xmin: The minimum value of the axis X

2) xmax: The maximum value of the axis X

3) ymin: The minimum value of the axis Y

4) ymax: The maximum value of the axis Y

5) component_list: A list of ID's of the fields to be plotted.
For example, to display the fields of ID 1 and ID 2, specify as [[1], [2]].
To display a sum of the fields of ID 1 and ID 2, and the field of ID 3,
specify as [[1,2], [3]].
The inner "[]" means that the values of the fields with the ID's enclosed

in this "[]" are summed up. Do not forget the inner "[]".
The default is empty, which means that all the fields are plotted.

6) name: Select the name of the field you want to display.

7) axis_name: Select the name of axis.

8) xstart: Start position in X or R axis (available in 2D and 3D)

9) ystart: Start position in Y or H axis (available in 2D and 3D)

10) zstart: Start position in Z axis (available in 2D and 3D)

6.5.2 show_field

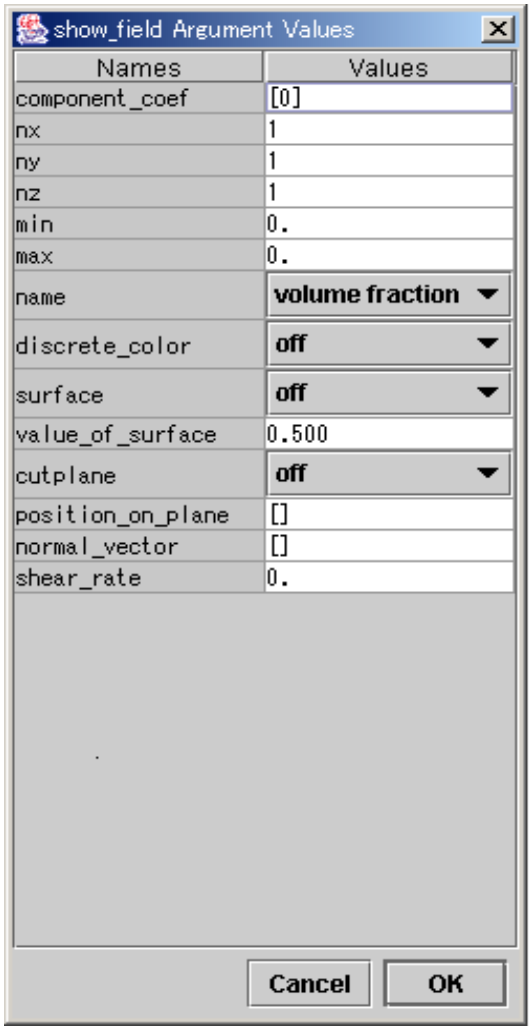


Figure 6.3: Parameters of the action box for "show field"

1) nx: Number of periodic of X axis

2) ny: Number of periodic of Y axis

3) nz: Number of periodic of Z axis

4) component_coef: The ID's of the fields to be plotted are specified with signs.

Replace the argument "0" with the ID of the field that you want to show.
If you specify a list of ID's instead of a single ID,
the sum of the fields for the listed species will be shown.
The negative value of ID means subtracting the value of the field.
For example, component_coef = [0, -1] means to show the value of
field[0] - field[1].

5) min: The minimum value of the range to be shown.

6) max: The maximum value of the range to be shown.
The default values are min = 0 and max = 0. This means that the range is set
automatically between the minimum value and the maximum value of the data.

7) name: Select the name of the field whose profile you want to display.

8) discrete_color: Show a multi component system using multi colors.

9) surface: Show interfaces of a three-dimensional system.
The interface is defined as the surface on which the density is constant.

10) value_of_surface: The value of surface to show.

11) cutplane: Show the density distribution on a cross section of a three-dimensional system.

12) position_on_plane: The position of the plane on which the crosssection is shown.

13) nomal_vector: Show the normal vector to the above plane for which the crosssection is drawn.

14) shear_rate: Shear rate, this value is inputted autonmatically in sheared dynamics, so no need to input.

Only one of the following switches can be effective: discrete_color, surface and cutplane.

6.5.3 value_of_record_plot

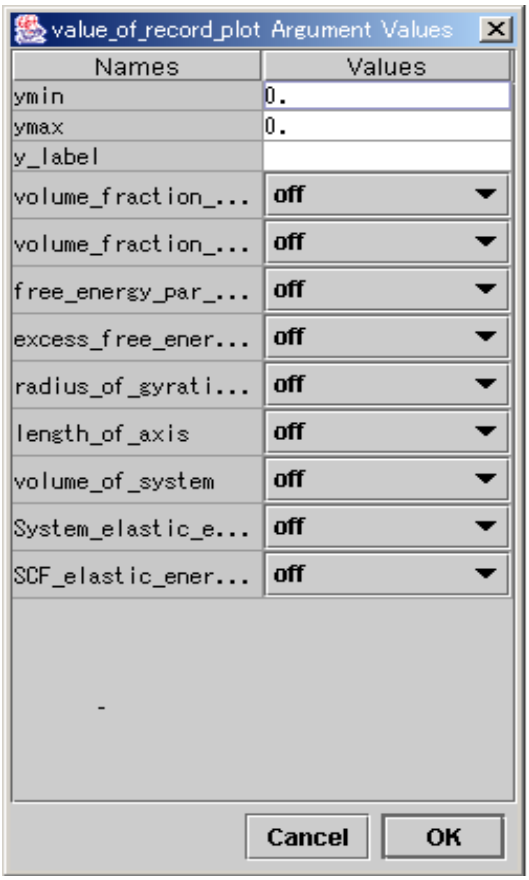


Figure 6.4: Parameters of the action box for "record plot"

The action file can be used to plot the values in each record. In the case of a dynamic calculation, the X axis is used to show the time. In the case of the SCF Monte Carlo simulation, X axis is used to show the number of steps of the record.

1) ymin: The minimum value of the Y axis.

2) ymax: The maximum value of the Y axis.
The default values are ymin = 0 and ymax = 0. This means that the range is set automatically between the minimum value and the maximum value of the data.

3) y_label: The label of the Y axis.
If this is not specified, the name of the selected variable is used.

4-12) values to plot: Toggle switches to specify whether the is drawn or not.

Chapter 7

CPC: Simple Python scripts for estimating the χ-parameters

7.1 Introduction

Although the χ-parameters are important parameters that determine the physical properties of polymer blend systems, such as the miscibility and the interfacial tension, it is difficult to estimate the values of the χ-parameters theoretically. One of the simple and well-known theoretical techniques to estimate the χ-parameters is the method using the solubility parameters.[25, 26, 27]

In the OCTA system, we offer you several UDF files that contain useful data on physical properties of various kinds of polymers. These UDF files are stored in the *POLYMERDATABASE* folder, and can be referred by using GOURMET. Hereafter, we call these UDF files as *PolymerDatabase*.

ChiParameterCalculator.py is a Python script that estimates the values of the χ-parameters using the data on the molar volume and the solubility parameter for the segments stored in *PolymerDatabase*.

If the data on the solubility parameter are not registered in *PolymerDatabase*, you can estimate them using the Python script named *SolubilityParameterCalculator.py*, which estimates the solubility parameters from the chemical structure of the monomer unit by using the group contribution method.

7.2 Principle of the method to estimate the χ-parameters

7.2.1 The Flory-Huggins lattice theory and the χ-parameters

In this section, according to the lattice theory on the polymer solutions/alloys proposed by Flory and Huggins, we define the Flory-Huggins interaction parameters, i.e. the so-called χ-parameters.[28]

In order to judge the miscibility or the solubility of a polymer solution or a polymer alloy, it is necessary to evaluate the mixing free energy, ΔG^M. This mixing free energy is defined as the difference in the free energy of the system between the states before and after the mixing, and is generally a function of the compositions, temperature, and molecular weights, etc. Assuming no volume change upon the mixing, the mixing free energy is written as

$$\Delta G^M = \Delta H^M - T\Delta S^M \tag{7.1}$$

in terms of the entropic contribution ΔS^M and the enthalpic contribution ΔH^M.

In order to obtain explicit expressions for each term in eq. (7.1), we use the lattice model. For simplicity, we consider a mixture of two homopolymers A and B in a system of volume $\mathcal{V}$. Each polymer is assumed to be composed of the same kind of segments, and we also assume that an A-segment and a B-segment have the same volume. The total numbers of the segments contained in an A-chain and in a B-chain are denoted as N_A and N_B, respectively. The contents of A-chains and B-chains in the system are assumed to be M_A and M_B moles, respectively. Using the theory of regular solution mixture, the total mixing entropy of the system, ΔS^M, is calculated in the same manner as the mixing entropy of an ideal gas. This is just counting the numbers of possible configurations of M_A A-polymer chains and M_B B-polymer chains on $M_A + M_B$ lattice sites, and gives the following expression.

$$\Delta S^M = -R(M_A \ln \phi_A + M_B \ln \phi_B), \tag{7.2}$$

where R is the gas constant. Here, ϕ_A and ϕ_B are the volume fractions of the A-polymer and the B-polymer, which are defined by

$$\begin{aligned} \phi_A &= \frac{M_A N_A}{M_A N_A + M_B N_B} \\ \phi_B &= \frac{M_B N_B}{M_A N_A + M_B N_B}. \end{aligned} \tag{7.3}$$

Equation (7.3) can be solved with respect to M_A and M_B to give

$$M_A = \frac{\mathcal{V}}{V_r}\frac{\phi_A}{N_A}, \qquad M_B = \frac{\mathcal{V}}{V_r}\frac{\phi_B}{N_B}. \tag{7.4}$$

Here, V_r is the molar volume of the segment, which is assumed to be the same for A and B. In the Flory-Huggins lattice model, these segmental molar volume V_r is equivalent to the volume assigned to a lattice site, i.e. a volume of a cube with its side length equal to the effective bond length b.

Substituting eq. (7.4) to eq. (7.2), the following relation is obtained for the mixing entropy.

$$\frac{-T\Delta S^M}{RT(\mathcal{V}/V_r)} = \frac{\phi_A}{N_A}\ln\phi_A + \frac{\phi_B}{N_B}\ln\phi_B. \tag{7.5}$$

In the Flory-Huggins lattice theory, all the complicated effects of mixing except for the mixing entropy of regular solution given by eq. (7.5) is accounted for by a single parameter χ_{AB}. Then, the mixing free energy is written as follows.

$$\frac{\Delta G^M}{RT(\mathcal{V}/V_r)} = \frac{\phi_A}{N_A}\ln\phi_A + \frac{\phi_B}{N_B}\ln\phi_B + \chi_{AB}\phi_A\phi_B \tag{7.6}$$

This expression is called as the Flory-Huggins mixing free energy and the parameter χ_{AB} is called as the Flory-Huggins interaction parameter, or simply called χ-parameter.

Due to the above definition of the Flory-Huggins free energy, the χ-parameter obviously includes not only the enthalpic contribution resulting from the contact interaction between segments but also the entropic contribution other than the mixing entropy of the regular solution (so-called combinatorial entropy). An example of the latter entropic contribution is the entropy associated with the internal degrees of freedom of the segments such as the relative orientation of two interacting segments. Although the contact interaction in eq. (7.6) is assumed to be proportional to the product of the segment densities, it is common that the actual contact interaction shows much complicated dependence on the segment density. For these reasons, the χ-parameter introduced in eq. (7.6) is in general a complicated function of the degree of polymerization, the density of the polymer chain, temperature, etc.

The specific volume of the K-type segment r_K, introduced in Section 2.9.2, is calculated using the molar volume V_K of the K-type polymer and the segment molar volume V_r as follows.

$$r_K = V_K/V_r \tag{7.7}$$

7.2.2 Solubility parameters

In order to calculate the contact interaction term in eq. (7.6), we assume that the contact interaction works only between the nearest neighbor lattice sites. Let us denote the number of the nearest neighbor lattice sites as z. Evaluating the contact interactions using mean field approximation, the χ-parameter in eq. (7.7) is expressed as follows.

$$\chi_{AB} = \frac{z}{k_{\mathrm{B}}T}\Big[\frac{1}{2}(\epsilon_{AA}+\epsilon_{BB}) - \epsilon_{AB}\Big] \tag{7.8}$$

Here, $\epsilon_{KK'}$ is the interaction energy between a nearest neighbor pair of a K-type segment and a K'-type segment. For pairs of non-polar segments with no long range interactions, it is a good approximation to assume that

$$\epsilon_{AB} \sim \sqrt{\epsilon_{AA}\cdot\epsilon_{BB}}. \tag{7.9}$$

Applying this approximation to eq. (7.8), the following expression is obtained.

$$\chi_{AB} = \frac{z}{2k_{\mathrm{B}}T}(\sqrt{\epsilon_{AA}} - \sqrt{\epsilon_{BB}})^2 \tag{7.10}$$

The interaction energy between a segment pair ϵ_{KK} in eq. (7.10) can be calculate using the values of the cohesive energy of pure systems. Here, the cohesive energy is defined as the energy required to take a cohesive state of molecules into isolated molecules with infinite separation. Let us consider a system of volume $\mathcal{V}$ composed of $\mathcal{N}$ lattice sites. If all the lattice sites are occupied by the K-type segments, the total interaction energy of this system is given by

$$E_K^{\rm coh} = -\frac{1}{2} z\mathcal{N}\epsilon_{KK} \equiv \mathcal{V}\delta_K^2. \tag{7.11}$$

Here, the parameter δ_K ($K = A$ or B) is called the solubility parameter, and is defined using the cohesive energy $E_K^{\rm coh}$ of the pure system composed of the K-type segments as

$$\delta_K = \Big(\frac{E_K^{\rm coh}}{\mathcal{V}}\Big)^{1/2}. \tag{7.12}$$

The total interaction energy of the uniformly mixed state of A and B segments with the volume fractions ϕ_A and ϕ_B, respectvely, is given under the mean field approximation as

$$\begin{aligned} E_{A+B}^{\rm coh} &= -\frac{1}{2} z\mathcal{N}\Big[\epsilon_{AA}\phi_A^2 + \epsilon_{BB}\phi_B^2 + 2\epsilon_{AB}\phi_A\phi_B\Big] \\ &= -\mathcal{V}\Big[\delta_A^2\phi_A^2 + \delta_B^2\phi_B^2 + 2\delta_A\delta_B\phi_A\phi_B\Big]. \end{aligned} \tag{7.13}$$

By using eqs. (7.11) and (7.13), the mixing enthalpy is written as follows.

$$\begin{aligned} \Delta H^M &= (E_{A+B}^{\rm coh} - E_A^{\rm coh}\phi_A - E_B^{\rm coh}\phi_B)/\mathcal{V} \\ &= (\delta_A - \delta_B)^2\phi_A\phi_B \end{aligned} \tag{7.14}$$

By using this relation and eq. (7.8), an expression for the χ-parameter in terms of the solubility parameter is obtained as

$$\chi_{AB} = \frac{V_r}{RT}(\delta_A - \delta_B)^2, \tag{7.15}$$

where, V_r is the molar volume defined in the previous section.

The other contribution to the interaction parameter except for the enthalpic contribution given in eq. (7.15) is mainly due to the entropic effect associate with the internal degrees of freedom of the segments, and is usually denoted as χ_S. By adding this term to eq. (7.15), the final expression of the χ-parameter is often expressed as

$$\chi_{AB} = \chi_S + \frac{V_r}{RT}(\delta_A - \delta_B)^2. \tag{7.16}$$

Here, χ_S expresses the contribution that cannot be expressed by eq. (7.15) and is empirically assumed to be [25]

$$\chi_S \sim 0.34. \tag{7.17}$$

In eq. (7.16) which is the final expression of the χ-parameter, the solubility parameters δ_A and δ_B are constants characteristic to the substances at the given temperature. On the other hand, there still remains an ambiguity in the segment molar volume V_r depending on the definition of the segment. Therefore, in order to use eq. (7.16), it is necessary to specify the segment molar volume V_r, or in other words, one has to define what is the segment. (Once the segment is defined, the total number of segments in a chain N_K is automatically given.)

7.2.3 The group contribution method

As was discussed in Section 7.2.2, one can estimate the value of the χ-parameter using eq. (7.16) when the values of the solubility parameter (or the cohesive energy) of pure systems are provided. Though the precise value of the solubility parameter should be determined in experiments, a rough estimate of the parameter can be obtained using the so-called group contribution method. In this technique, the solubility parameter of a monomer unit is expressed as a sum of independent contributions from the atomic groups that constitute the monomer unit.[25]

For example, the monomer unit shown in fig. 7.1 can be divided into the following groups.[25] The

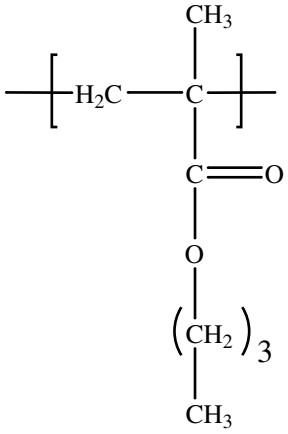


Figure 7.1: n-butyl methacrate monomer

values of the contributions to the cohesive energy from atomic groups have been estimated by various researchers.[25, 26, 27]

Group	Number
$-CH_2-$	4
$-CH_3$	2
$>C<$	1
$-COO-$	1

For example, if we use the values proposed by Hoftyzer and van Krevelen (1976)[29], the cohesive energy can be estimated as follows.

Group	Number	CohesiveEnergy(J/mol)
$-CH_2-$	4	4190
$-CH_3$	2	9640
$>C<$	1	-5580
$-COO-$	1	13410

$$\text{Sum} \quad E^{\text{coh}} = 4 \times 4190 + 2 \times 9640 + 1 \times (-5580) + 1 \times 13410 = 43870(\text{J/mol}) \tag{7.18}$$

Then, as the molar volume of this monomer unit is $V_r = 136$ (cm^3/mol), the solubility parameter can be estimated as

$$\delta = \sqrt{E^{\text{coh}}/V_r} = 18.0 \quad (\text{J}^{1/2}/\text{cm}^{3/2}). \tag{7.19}$$

When the value of the molar volume V_r is unknown, it can be estimated by using the method of Fedors[30]. In this method, in a similar manner used for the cohesive energy, the value of the molar volume for a segment is calculated using the values of the molar volume of the constituent groups. For the above-mentioned example, the molar volume is calculated as follows.

Group	Number	MolarVolume(cm^3/mol)
$-CH_2-$	4	16.1
$-CH_3$	2	33.5
$>C<$	1	-19.2
$-COO-$	1	18.0

$$\text{Sum} \quad V_r = 4 \times 16.1 + 2 \times 33.5 + 1 \times (-19.2) + 1 \times 18.0 = 130.0(\text{cm}^3/\text{mol}) \tag{7.20}$$

The calculated value is close to the experimental value $V_r = 136$ (cm^3/mol).

7.3 Operation of various scripts

7.3.1 ChiParameterCalculator : χ-parameter estimation using PolymerDatabase

The values of many physical properties of various monomers are stored in *PolymerDatabase*. In this *PolymerDatabase*, the data of the solubility parameter (in unit of $J^{1/2}/cm^{3/2}$) and the molar volume (in unit of cm^3/mol) are also contained. The Python script *ChiParameterCalculator.py* is a script to estimate the χ-parameters using these data.

The procedure to use *ChiParameterCalculator.py* is as follows.

(1) Start GOURMET and open *polymerdata.udf* in *PolymerDatabase* by using `UDF Editor`.

(2) Set Table-View, and select **PolymerDatabase** → **GeneralPolymers**[] → **Properties**[] .

(3) Among the many items, *Flag:int* is the only item that you need to edit.

The **PolymerDatabase.GeneralPolymers**[] on the upper level of the data structure shows the correspondence between the data column and the monomer species. Enter non-zero integers in the *Flag:int* items of the monomers for which you want to calculate the χ-parameters. (You can specify more than two monomers. In such a case, χ-parameters for all the possible pairs are calculated.)

(4) Click **Load** in the Python-panel, and load

SUSHI3/ChiParameterCalculator/python/ChiParameterCalculator.py .

(5) In the Python Scripting Window, find the following lines. (They are located between 28th and 30th lines.)

```
#############################################################################
    Vr          = -1.0E99   # Please enter molar volume
                            #   of the segment in ( cm^3 / mol ).
    T           = -1.0E99   # Please enter absolute temperature in ( K ).
#############################################################################
```

Replace each -1.0E99 with the value of the molar volume in cm^3/mol and the temperature value in Kelvin. If you do not specify these values, the Python script will be aborted.

(6) Click **Run** in the Python-panel to execute this script.

(7) The result will be shown in the Python Log Window. (χ-parameters are dimensionless.)

Note 1 : You should execute the script after substituting the values of the molar volume V_r (in unit of cm^3/mol) of the segments and the absolute temperature T (in unit of K) in the Python script. As was described in Section 7.2.2, the molar volume of a segment is ambiguous depending on its definition. Therefore, the definition of the molar volume and the polymer chain length N_K should be selected consistently.

Usually it is acceptable to use the value $V_r = 100$ (cm^3 / mol), which is a typical value of the molar volume of low molecular solvents, such as toluene, etc. Once you define the segmental molar volume, the total number of segments per chain N_K is calculated according to the following relation.

$$\begin{aligned} N_K &= \text{(total number of segments in the chain)} \\ &= \text{(total number of monomers in the chain)} \times \text{(molar volume of the monomer)} \\ &\quad /\text{(molar volume of the segment } V_r\text{)} \\ &= \{\text{(molecular wegiht of the chain)} \times \text{(molar volume of the monomer)}\} \\ &\quad /\{\text{(molecular weight of the monomer unit)} \times \text{(molar volume of the segment)}\}. \end{aligned} \quad (7.21)$$

Therefore, if the molar volume V_r of the segment is given, the total number of segments N_K which constitute the K-type chain can be calculated using this method. On the other hand, another condition is required to

determine the effective bond length b_K. For example, if the value of the radius of gyration R_G of the chain is provided, the effective bond length b_K can be determined as

$$\frac{1}{6}N_K b_K^2 = R_G^2. \tag{7.22}$$

According to eq. (7.7), the specific segment volume of the K-type segment r_K is determined as

$$r_K = V_K/V_r. \tag{7.23}$$

Note 2 : When the value of the solubility parameter is -1.0E99 in `PolymerDatabase`*, it means that the corresponding data is not registered. The execution of the Python script using such data will result in a failure with the following message.*

```
================================================
  The value(s) of the solubility parameter is(are)
  not defined.
  Calculation aborted.
================================================
```

In this case, it is necessary to register the value of the solubility parameter into `SolubilityParam:double` *(in unit of $J^{1/2}$ / $cm^{3/2}$). When the value of the solubility parameter is unavailable, it can be estimated using the group contribution method as will be described in Section* `SolubilityParameterCalculator`*.*

7.3.2 SolubilityParameterCalculator : Estimating the Solubility Parameters Using the Group Contribution Method

When the data of the solubility parameter are unavailable, these can be estimated using the group contribution method. For this purpose, we provide several databases (UDF files) in *PolymerDatabase* which store the numerical data used in the calculations based on the group contribution method. The user should choose one of the following databases.

- SolubilityParameter_Hoftyzer&vanKrevelen.udf :
  The original data can be found in the reference [29] except for the data of the molar volume which can be found in the reference [30].
  (Note) Although the accuracy of the data is better than the other databases, the numbers of the registered groups are less than those of the others.

- SolubilityParameter_Dunkel.udf :
  The original data can be found in the reference [31] except for the data of the molar volume which can be found in the reference [30].
  (Note) The numbers of the registered groups are comparable with those of
  `SolubilityParameter_Hoftyzer&vanKrevelen.udf`*, and the accuracy of the data is also on the same level.*

- SolubilityParameter_Fedors.udf :
  The original data can be found in the reference [30].
  (Note) The numbers of the registered groups are many but the accuracy is rather worse.

In order to estimate the solubility parameter using these databases, a Python script *SolubilityParameterCalculator.py* is provided.

The procedure to use *SolubilityParameterCalculator.py* is as follows.

(1) Start GOURMET and open *PolymerDatabase* by using `UDF Editor`.

The file name of the database is one of the followings,
POLYMERDATABASE/SolubilityParameter_Hoftyzer&vanKrevelen.udf ,
POLYMERDATABASE/SolubilityParameter_Dunkel.udf,

or
POLYMERDATABASE/SolubilityParameter_Fedors.udf.

(2) Set Table-View and select **SolubulityParameterDatabase** → **GeneralProperties**[] .

(4) Among many items, *NumberOfUnits:int* is the only item you need to modify.
Each column corresponds to the group specified in *GroupName*. You need to set the number of the groups in the target monomer into *NumberOfUnits:int*.

(5) Click **Load** in the Python-panel to load the file named as
SUSHI3/ChiParameterCalculator/python/SolubilityParameterCalculator.py.

(6) Click **Run** in the Python-panel to execute the script.

(7) The results (the values of the cohesive energy and the molar volume) are shown in the Python Log Window.

Note 1 : Since the values of the solubility parameter calculated by `SolubilityParameterCalculator.py` *are not automatically registered to the* `PolymerDatabase`*, the user has to register the obtained values by yourself to* `PolymerDatabase` *or some other UDF files.*

Note 2 : When either of `CohesiveEnergy:double` *or* `MolarVolume:double` *in* `SolubilityParameterDatabase_*****.udf` *is specified as -1.0E99, it means that the data of the cohesive energy or the molar volume of this group is not registered. If you execute the script in this case, you will get no calculation result. Instead, the following message will be displayed.*

```
================================================
  The value(s) of the solubility parameter is(are)
  not defined.
  Calculation aborted.
================================================

================================================
  The molar volume of the segment Vr
    or the temperature T is not defined.
  Calculation aborted.
================================================
```

Chapter 8

SPCF: A tool for calculating the spatial correlation functions

Spcf *.py file was moved as action file. Please use SPCF button shown by right-click of mouse button on SUSHIOutput sub-holder.

8.1 Outline

As a result of the simulation using SUSHI, we obtain scalar field data such as the segment density distributions, which show the spatial distributions of the polymers. Spatial correlation function $\langle\phi(0)\phi(r)\rangle$ is one of the measures of the spatial order in the scalar field $\phi(\mathbf{r})$. To obtain this quantity, an auxiliary analysis tool SPCF is provided. A graph of the spatial correlation functions can be obtained by using SPCF.

Here, the spatial correlation function $\langle\phi(0)\phi(r)\rangle$ is defined as following

$$\langle\phi(0)\phi(r)\rangle = \frac{\sum_{i,j\in|\mathbf{r}_{ij}|=r} \phi_i\phi_j}{\sum_{i,j\in|\mathbf{r}_{ij}|=r} 1}, \tag{8.1}$$

where, i and j are indices to specify the positions, ϕ_i and ϕ_j are the values of the target scalar fields at positions i and j, and $\mathbf{r}_{ij}$ is the relative position vector from i to j, respectively. Since the scalar fields are discretized in the numerical computations, the distance r is also discretized. Therefore, in the calculation of SPCF, a suitable value for the unit length Δr should be set. Then, the distance r is defined by

$$|\mathbf{r}_{ij}| = r \qquad \Longrightarrow \qquad r - \frac{\Delta r}{2} \le |\mathbf{r}_i - \mathbf{r}_j| \le r + \frac{\Delta r}{2}, \tag{8.2}$$

where $\mathbf{r}_i$ and $\mathbf{r}_j$ are the position vectors.

8.2 How to execute

The SPCF is an executable program that can be operated in the command prompt. It requires a command file that stores the operations directed to the SPCF at run-time and the data file that stores the target density fields for the analysis. Suppose that these files are prepared, the operations in the command prompt should be as follows.

```
% spcf.exe ctrl.cmd
```

In this example, the file names of the executable program and the command file are `spcf.exe` and *ctrl.cmd*, respectively. In the software media of OCTA system, the executable files of the SPCF for WindowsNT/2000 (cygwin) and Linux (x86) are stored in the *SUSHI3/spcf/bin* directory.

[Notes] Although SPCF is a stand-alone program, the executable file for cygwin requires a dynamic link library (cygwin1.dll) at run time. Thus, you have to copy cygwin1.dll to the same directory as that contains the executable file or to one of the directories registered in the environment variable `PATH`.

Each line in the command file consists of a keyword ending with colon (:) followed by one or more values that are separated with one or more blank characters (spaces or tabs) with each other. here, the keyword is one of the following strings:

'rmax:', 'rstep:', 'spherical:', 'dirvec:', 'clip:', 'shift:', 'pairlist:', 'infile:', 'outfile:', 'outfile-format:', and the value is one or more numerics or strings, which depend on the corresponding keyword.

An example of the syntax of the command file is shown below.

```
# input data for 'spcf.exe'
# The lines starting with '#'-character are the comment lines,
# and are neglected.
#  Blank lines (such as the next line) are also neglected.

# The upper bound of the distance up to which the calculation is performed.
# [format] rmax: (numeric)
rmax: 16.0

# The mesh width for the distance
# [format] rstep: (numeric)
rstep: 1.0


# In case the keyword 'spherical:' is false,
# the distance is defined as the size of the projection of the relative
# position vector to the direction specified by the keyword
# dirvec:
# spherical: false
# dirvec:  1.0 1.0 1.0


# The upper bound and the lower bound of the field data.
# The field values outside of this range are replaced by the upper bound value
# or the lower bound value.
# [format] clip: (lower-limit) (upper-limit)
clip:  0.0  1.0

# The values by which the field data for each component are shifted.
# The number of the values should be equal to the
# number of the components. Each of the values is added to the field data
# of each component.
# [format] shift: (numeric) (numeric) ...
shift: -0.2  -0.8


# The type of the component pair for which the correlation function is
# calculated.  "self", "distinct" and "all" mean the self-correlation,
# the correlation function between distinct species, and the correlation
# function for all possible pairs of the components, respectively.
# [format] pairlist: self | distinct | all
pairlist: self

# The name of the file that stores the input data of the scalar field.
# [format] infile:  filename
infile: Susi_Phi.dat

# The name of the file to which the output data are written
# and the data format of the outfut file.
# The type of the data format is one of GNUPLOT-style(gnuplot),
# tab separated style(tabtext), or comma separated style (csvtext).
# In the case of GNUPLOT-style, the command file for GNUPLOT is also
# created. The command file is named as XXX.gnuplot.
# [format] outfile: filename
# [format] outfile-format:  gnuplot | tabtext | csvtext
outfile:  plot.dat
outfile-format: gnuplot
```

Each line in the input data file of the density fields consists of the x, y and z-coordinates of the position vector and the field value(s). An example of the syntax of the input file is shown in the following.

```
#  scalar field data
# The lines starting with '#'-character are comment lines, and are neglected.
#
#  X-value    Y-value    Z-value     phi-0       phi-1
   0.0        0.0        0.0         0.199685    0.800365
   0.0        0.0        0.5         0.200342    0.801061
   0.0        0.0        1.0         0.201941    0.798672
   0.0        0.0        1.5         0.20148     0.801061
   0.0        0.0        2.0         0.20088     0.801031
# In this example, the number of the fields is 2.
# As long as the data are described in a single line,
# any number of the fields are allowed.
```

The result of the calculation is stored in another file, whose name and format are specified in the command file. To assist the users to create the input data file of the density field(s) from the output data of SUSHI, a Python script `SUSHI2spcf.py` is provided. You can find this script in the `SUSHI3/spcf/python` directory. The usage of this script is shown in the following.

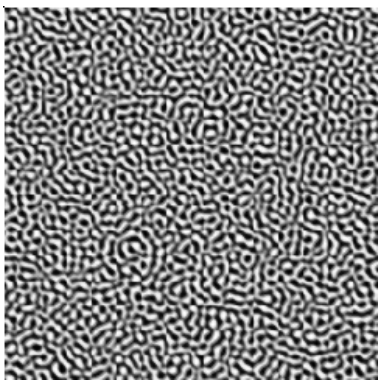
1. Start GOURMET and load the UDF output of SUSHI by `Browser` or `Editor`.
2. Set Table-View and Record-Location, and change the slider to set the target record number.
3. Push **Load** in the Python panel to load the file named *SUSHI2spcf.py*
4. Push **Run** in the Python panel to create the output data file for the density field that can be read by SPCF. You may change the file name or the folder name by modifying "destdir" (180th line) and "datafile" (177th line) in the *SUSHI2spcf.py*.

8.3 Sample data

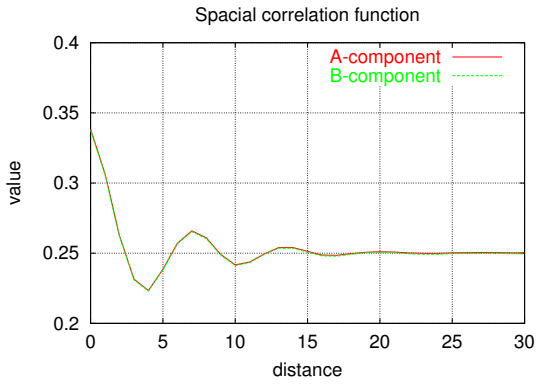
Several samples of SPCF are provided in the *SUSHI3/spcf/sample* directory.

8.3.1 A/B polymer blend

The density filed obtained by performing SUSHI for a blend system of polymer A and polymer B is shown in fig. 8.1a. Corresponding spatial correlation function is shown in fig. 8.1b. The simulation system is a 2-dimensional square box with 256×256 meshes, and the volume fraction ratio, the chain length, and the interaction parameters of the polymers are 0.5:0.5, 10 for A-polymer and 10 for B-polymer, and $\chi_{AB} = 0.5$, respectively. In fig. 8.1a, the density of the polymer A is shown by gray scale, black corresponding to 0.0 and white to 1.0. The names of the files used/obtained in this example are *A-B-blend.cmd*, *A-B-blend.dat*, and *A-B-blend.plot*.



(a) Segment density distribution

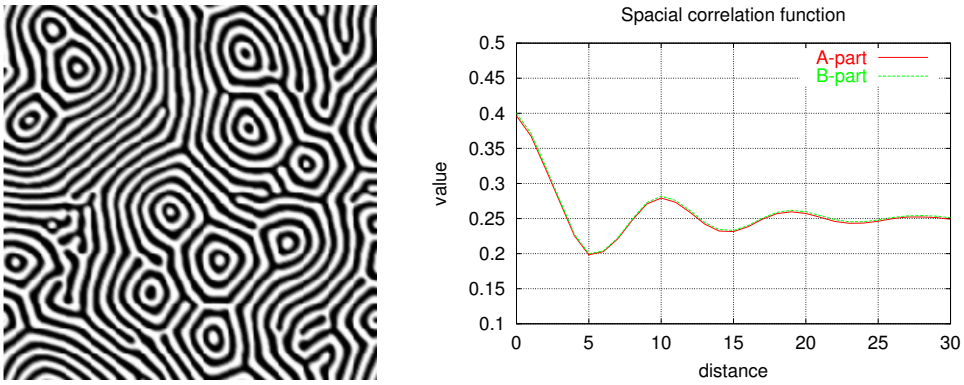


(b) Spatial correlation function

Figure 8.1: The segment density distribution and the spatial correlation function of an A/B polymer blend

8.3.2 A-B block polymer

The density filed obtained by performing SUSHI for an A-B block copolymer melt system is shown in fig. 8.2a. Corresponding spatial correlation function is shown in fig. 8.2b. The simulation system is a 2-dimensional square box with 256×256 meshes, and the chain length and the interaction parameters of the polymers are 20 for the A-block and 20 for the B-block, and $\chi_{AB} = 0.5$, respectively. In fig. 8.2(a), the density of segment A is shown by gray scale, black corresponding to 0.0 and white to 1.0. The names of the files used/obtained in this example are *A-B-block.cmd*, *A-B-block.dat*, and *A-B-block.plot*.



(a) Segment density distribution

(b) Spatial correlation function

Figure 8.2: The segment density distribution and the spatial correlation function of an A-B block copolymer melt

8.3.3 Effect of the direction vector

Generally, the correlation function is obtained as the spherically averaged value over all the displacement vectors with a fixed magnitude r. However, when the system has the anisotropy toward a certain direction, the structure may become clearer if the distance r is defined as the projection onto the direction. Such an example, where a periodicity exists in the (1,1,0)-direction, is shown in fig. 8.3.

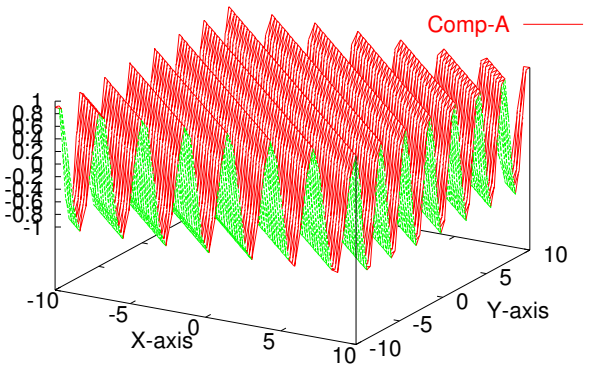
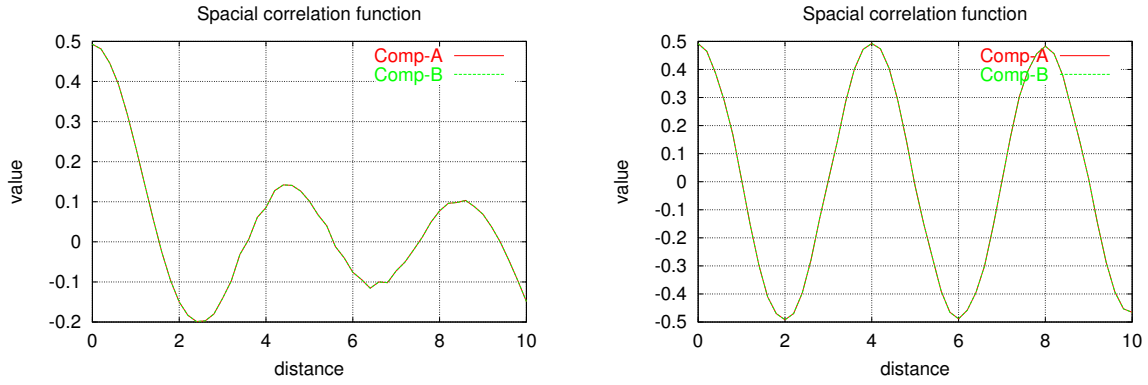


Figure 8.3: A result with the periodicity in the (1,1,0)-direction: $z_A = cos((x+y) \times \frac{\pi}{\sqrt{2}})$, $z_B = -z_A$

The utility "spcf" has a function to calculate such a correlation function for the anisotropic systems where the distance is defined as the projected size of the relative vector toward the direction of the anisotropy. In such a case, the keyword **spherical:** should be set as `false` and the direction vector should be specified by

using the values of keyword **dirvec:**. An example of anisotropic density field is given in fig. 8.3. For such a density field, the circularly averaged correlation functions and the anisotropic correlation function to the specific direction are calculated and the results are shown in figs. (8.4a) and (8.4b), respectively. The names of the files used/obtained in this example are *dirvec.cmd*, *dirvec.dat*, and *dirvec.plot*.



(a) Usual spatial correlation function

(b) Spatial correlation function projected to the (1,1,0)-direction

Figure 8.4: Results of (a) the circularly averaged correlation function and (b) the anisotropic correlation function to the specified direction for the density field shown in fig. 8.3.

Appendix A

Input UDF format for navigating data input

SUSHIInput.udf defines the input data structure for SUSHI. Using this UDF format, one can prepare the input UDF data conveniently. However, the way in which GOURMET displays the input data structure is not so intelligent. In order to improve this problem, we prepared SUSHIInputV2.udf where the select function of UDF is extensively used to assist the user to type the data on GOURMET. Since most of the definitions in SUSHIInputV2.udf is the same as those in SUSHIInput.udf, such common part will not be described in the following. Basic data structures defined in SUSHIInputV2.udf are as follows.

```
SUSHIInputV2:{
    System:{
      name:KEY
      mesh:select { "REGULAR", "RECTANGULAR", "CYLINDRICAL", "SPHERICAL" }
    }
    Monomers[]:Monomer
    Components:{
        polymers[]:Polymer
        solvents[]:Solvent
    }
    Chi_parameters[]:ChiParameter
    Ensemble: {
        type:select { "CANONICAL", "GRANDCANONICAL" }
    }
    Properties:{
        segment_volume_fraction_conditions[]:SegmentVolumeFractionCondition
        radius_of_gyration_conditions[]:SubchainUnit
    }
    External_conditions:{
        surface_chi_parameters[]:SurfaceChiParameter
        graft_conditions[]:GraftCondition
        mask_conditions[]:MaskCondition
    }
    Solver: {
     type:select { "ADF", "FH", "SCF" }
    }
    Run: {
        type:select { "START", "CONTINUE", "RESTART", "RESTART_READMESH" }
    }
}
```

The order of the above data are different from that defined in SUSHIInput.udf.

`Sysytem`, `Components`, `Chi_parameter`, `Ensemble`, `Properties`, `External_condition`, `Solver`, and `Run`. The select function of UDF navigates the user to correct items that should be specified, which prevents the mistakes from taking place. For example, Regular mesh can be specified as one, two and three-dimensional structures, while Spherical mesh is allowed to be only one-dimensional. In such a case, the selected function prevents the user from setting a wrong combination of the Mesh type and the dimensionality by locking the items that should not be specified.

```
System:
  mesh:select
    +-REGULAR:
        dimension:select
          +-D1 // one-dimensional
          +-D2 // two-dimensional
          +-D3 // three-dimensional
    +-RECTANGULAR:
        dimension:select
          +-D1
          +-D2
          +-D3
    +-CYLINDRICAL:
        dimension:select
          +-D2
    +-SPHERICAL:
        dimension:select
          +-D1
Ensemble:
  type:select
    +-CANONICAL:
        calculation_method:select
          +-STATICS
              external_conditions_of_statics:ExternalConditionsOfStatics
          +-DYNAMICS
              dynamics_parameter:DynamicsParameter
              external_conditions_of_dynamics:ExternalConditionsOfDynamics
          +-MONTECARLO
              monte_carlo_parameter:MonteCarloParameter
              external_conditions_of_monte_carlo:ExternalMonteCarlo
    +-GRANDCANONICAL:
        calculation_method:select
          +-STATICS
              external_conditions_of_statics:ExternalConditionsOfStatics
Solver:
  type:select
    +-ADF
    +-FH
    +-SCF
Run:
  type:select
    +-START    // start new calculation
    +-CONTINUE
    +-RESTART
    +-RESTART_READMESH
```

Although it is user's freedom to choose one of SUSHIInput.udf, SUSHIInputV2.udf, and SEED formats, SUSHIInputV2.udf format is recommended when there are no reference data file of SUSHIInput.udf format.

Appendix B

Compiling SUSHI

B.1 The directory structure for the source files

The directory structure for the source files of SUSHI is as follows.

```
SUSHI10.54--+--SurfaceSimulator
            |
            +--Susi--+--def_udf--+--SUSHIinput.udf   Input UDF definition file
                     |           +--SUSHIOutput.udf  Output UDF definition file
                     |
                     +--bin      The directory containing executable modules.
                     |
                     +--include  The directory containing include files.
                     |
                     +--src      The directory containing source files.
```

B.2 How to compile SUSHI on UNIX system

On the UNIX operating system (including cygwin on Windows), the GNU "make" command should be used for the compilation of the source codes. The correct compilation is guaranteed only for the GNU make command. To check whether your "make" command is GNU make command or not, use "make -v" command. If you are using the GNU make command, you will get like the following message.

```
> make -v
> GNU Make version 3.77, by Richard Stallman and Roland McGrath.
> ......
```

Make on SUSHI directory.

```
> cd SUSHI10.54
> make all
```

When you execute the "make all" command, all the libraries are first compiled and stored under the SUSHI/lib directory. Next, SUSHI and InterfaceSimulator are compiled using these libraries. The selection of the operating system (OS) is done automatically.

When the Makefile of SUSHI is not suitable for your environment and the compilation of SUSHI is failed, try to generate a Makefile as

```
> cd Susi/src
> sh ./mkmk.sh
```

these commands will make a new Makefile for your environment.

To add version information after module name, use VERSION option as

```
> make all [VERSION=version_number_etc]
```

If you want to compile only 1 core version SUSHI.

```
> cd SUSHI/susi/src
> make all
```

It is strongly recommended to use FFTW library when make SUSHI for RPA calculation. The option is FFTW=ON.

B.3 How to compile SUSHI for parallel computation

Pthread version

To make the shared memory version using pthread, use PTL option as

```
> cd SUSHI10.54/Susi/src
> make all PTL=ON
```

You can make sushiPTL.

GPU version

To make the shared memory version using CUDA, use GPU option as

```
> cd SUSHI10.54/Susi/src
> make all GPU=ON COMPUTECAPABILITY=60
```

You can make sushiGPU. Before this make, You need to install CUDA and GPU device and modify lines about CUDA in src/Makefile. COMPUTECAPABILITY will be selected for your GPU device. Default value is 60.

MPI version

To make the distributed shared memory and CPU version using MPI, use MPI option as

```
> cd SUSHI10.54/Susi/src
> make all MPI=ON
```

You can make sushiMPI. Before this make, You need to install some MPI library and modify lines about MPI in src/Makefile.

For K-computer, use the option MACHIN=K.
MPI and GPU, and MPI and PTL options are compatible.
We will use the version description as follows
MPU=MPI + GPU, MPR=MPI + PTL.

How to make all version

To make all versions incluiding 1 core version, make on SUSHI10.54 directory as

```
> cd SUSHI10.54
> make allsushi [VERSION=version_number_etc]
```

B.4 Building SUSHI with Microsoft Visual C++

In order to compile SUSHI with Microsoft Visual C++ (VC++), start Microsoft VisualStudio and load the project file named "susi.sln" on the SUSHI10.54/Susi directory. Then, build the project named "susi".

To build each parllel versions, these definitions in Susi/include/definitions.h should be enable.
//#define MPIUSE
//#define CUDAUSE
//#define PTHREADUSE

For Linux/UNIX version, these commentouts are not allowed.

Windows environment for VC++

The paths of libraries should be defined in the Windows's environment variables for VC++ as followings.

- libplatrom:PF_FILES(/include or /lib)
- MPI:MPI_INC_PATH, MPI_LIB_PATH
- CUDA:CUDA_PATH(/include, /lib)
- Pthread:PTHREAD_INC_PATH, PTHREAD_LIB_PATH
- FFTW:FFTW_INC_PATH, FFTW_LIB_PATH

For your build, no need libraries should be deleted in project file when build.
Or on SUSHI10.54 directory, type command

```
> .\buildall2017
```

which build all executable modules by VC++2017.

B.5 How to install SUSHI

In the case of UNIX-type OS, the following command install all executable moduldes to PF_ENGINE/bin/*

```
> cd SUSHI10.54
> make install
```

On Windows OS, the executable module stored is Susi/bin or in InterfaceSimulator/bin directories should be copied manually.

B.6 How to clean up the directories

In the case of UNIX-type OS, the temporary object files under SUSHI10.54 directory can be deleted by the following command.

```
> cd SUSHI10.54
> make clean
```

Appendix C

Extension of the system

C.1 How to write a program code for a static equilibrium simulation

SUSHI is not a complicated and mysterious program. It is build using many well-organized components (classes of C++). The class SCFEngine is the main part of the program. If you understand how to use this class, you could implement the SCF calculation performed by SUSHI into other programs. As an example, let us make a program for the static equilibrium calculation of a one-dimensional A/B polymer blend system. The program is as follows.

```
#include "SCFEngine.h"    //  Header file of the main part of the simulator.
#include "FieldNoise.h"   //  Header file of the generator of noise given to fields.

int main() {

    try {

        double sigma                   = 0.0001; // The standard deviation of the noise
                                                 // given to a field.
        int    randomSeed              = 12345;  // The seed for a random number.

        double minimumOfXAxis          =  0.;
        double maximumOfXAxis          = 32.;
        int    numberOfDivisionOfXAxis = 32 ;

        double deltaS                  = 1.;
        double effectiveBondLength     = 1.;
        double specificVolume          = 1.;
        double chainLength             = 20.;
        int    numPolymer              = 2;
        double chiAB                   = 0.2;
        double volumeFractionOfA       = 0.5;
        double volumeFractionOfB       = 0.5;

        // Monomers are defined first.
        Monomer monomerA( "A", specificVolume, effectiveBondLength );
        Monomer monomerB( "B", specificVolume, effectiveBondLength );

        // Polymers are made of monomers.
        Polymer homoPolymerA( monomerA, chainLength );
        Polymer homoPolymerB( monomerB, chainLength );
```

```
// Geometrical boundary conditions specified as the periodic boundary
// condition. The default is the periodic boundary condition.
GeometricalBoundaryCondition geometricalBoundaryCondition;

// Regular mesh is specified.
RegularMesh mesh(  "test", geometricalBoundaryCondition
                 , minimumOfXAxis, maximumOfXAxis
                 , numberOfDivisionOfXAxis );

// Physical boundary conditions on both ends are specified as
// an absorbing wall and a reflective wall. The default is
// the periodic boundary condition.
PhysicalBoundaryCondition physicalBoundaryCondition;

// The values of the chi parameters are specified.
ChiParameter chiParameter;
chiParameter.set("A", "B", chiAB );

// A container for the polymers is prepared using the STL vector class.
vector<Polymer*> polymers;
polymers.push_back( &homoPolymerA );
polymers.push_back( &homoPolymerB );

// A container for the values of the volume fractions of each component
// is prepared.
vector<double> polymerVolumeFraction;
polymerVolumeFraction.push_back( volumeFractionOfA );
polymerVolumeFraction.push_back( volumeFractionOfB );

// Containers for the scalar fields phi and fV are prepared.
// The values of the volume fractions and the self-consistent fields
// conjugate to them are stored.
vector< ScalarField* > phi, fV;
phi.push_back( new ScalarField( mesh, volumeFractionOfA ) );
phi.push_back( new ScalarField( mesh, volumeFractionOfB ) );
for( i = 0; i < numPolymer; i++ ) {
    fV .push_back( new ScalarField( mesh, 0. ) );
}

// To start the SCF calculation, small fluctuations are given as the
// initial value of one of fV's.
*fV[ 0 ] += FieldNoise( mesh, sigma, randomSeed );

// Constract the simulation engine.
SCFEngine engine(  polymers
                 , polymerVolumeFraction
                 , chiParameter
                 , mesh
                 , physicalBoundaryCondition
                );

// The function getEquilibrium is called.  The calculation results are
// stored in phi and fV.
cout << "nSCF " << engine.getEquilibrium( phi, fV ) << endl;

// Output phi to the console.
for( i = 0; i < numPolymer; i++ ) {
```

```
            cout << "phi " << i << " " << *phi[i] << endl;
        }

        // Delete all objects that are generated by "new" command.
        deleteFields( phi );
        deleteFields( fV );

    }

    catch( MFException x) {
        x.display(cerr);  //  In case an error occurs, a message is displayed.
    }

    return 0;
}
```

Such a simple program really runs. To controll the simulation, one can add an instance of the class SCFChar argument as an argument of the "getEquilibrium" function. The default conditions prepared in the SCFChar are enough to solve rather simple problems. Following is an example of the calculation result.

```
.....
nSCF 388
phi 0
 0.022021319 0.023472855 0.02768036 0.039853661 0.076080462
 0.18612184 0.45549125 0.76249782 0.90618754 0.95434671
 0.97035272 0.97586031 0.9777627 0.97839868 0.97856398
 0.97846946 0.97799456 0.97654264 0.97232646 0.96013022
 0.92385038 0.81371286 0.54438197 0.23758755 0.093894333
 0.045705149 0.029682295 0.024167216 0.022258887 0.021616024
 0.021445561 0.021542225

phi 1
 0.97798162 0.97653571 0.97225781 0.96010011 0.92395215
 0.81397296 0.54452043 0.23743126 0.09379216 0.0456568
 0.02965645 0.024150693 0.022248863 0.021613048 0.021447831
 0.021542434 0.022017497 0.023469804 0.027687145 0.039888208
 0.076186746 0.18636283 0.45562929 0.76235723 0.90609141
 0.95424992 0.9702444 0.97582681 0.97774844 0.97837143
 0.97855848 0.97845004
```

Finally the values of the field phi are displayed. You can confirm that a phase-separated structure is successfully simulated. We hope this simple example gives you an idea on how to perform SCF calculations using SCFEngine.

C.2 How to write a program code for a dynamic mean-field simulation

The calculations that can be performed by SUSHI are not restricted to the static equilibrium calculations. It can also perform dynamic mean-field simulations,

The class SCFEngine is the core program of SUSHI that performs the SCF calculations. It is a derived class of a virtual class named MeanFieldEngine. MeanFieldEngine has many interfaces with which one can get information on the system. One can construst a new engine for the mean-field simulation using this MeanFieldEngine class as a base class. Then, let us build a dynamic mean-field simulator based on the

Cahn-Hilliard type equation of motion, and apply it to the phase separation of an A/B polymer blend as an example. A basic equation for this system is as follows.

$$\frac{\partial}{\partial t}\phi_K(\mathbf{r},t) = \nabla[L_K(\mathbf{r},t)\nabla\mu_K(\mathbf{r},t)]. \tag{C.1}$$

The chemical potential μ_K is derived from the following extended Flory-Huggins free energy model (See eq. (2.35))

$$\frac{\mathcal{F}}{k_{\rm B}T} = \sum_K \frac{\phi_K}{N_K}\ln\phi_K + \sum_K \sum_{K'>K} \chi_{KK'}\phi_K\phi_{K'} + \sum_K \frac{\kappa}{2}|\nabla\phi_K|^2, \tag{C.2}$$

where κ is a positive coefficient that is related to the energy accumulated at the interface.

Let us construct a simulation engine for the above dynamical model. Actually, such a simulation engine has already been implemented in SUSHI with the name `FHEngine`. Please take a look at the source program for more detail. The contents of the header file are as follows.

```
#ifndef _FHENGINE_H_
#define _FHENGINE_H_

#include "MeanFieldEngine.h"
#include "FHChar.h"

class FHEngine: public MeanFieldEngine  {
public:
  FHEngine( .....    // Constructor
            .....
          );
  int getEquilibrium
        (  vector<ScalarField*>& phi, vector<ScalarField*>& fV
         , vector<int> isConstrained = vector<int>(0)
         , MeanFieldChar* pMeanFieldChar = NULL
        ) ;
  int getChemicalPotential
        (  const vector<ScalarField*>& phi, vector<ScalarField*>& fV
         , MeanFieldChar* pMeanFieldChar = NULL
        );
};

#endif //_FHENGINE_H_
```

As the program is derived from `MeanFieldEngine`, an explicit definition of the constructor is not neccesary. The function `getChemicalPotential` is to calculate the chemical potential using an appropriate model of the free energy. As this function `getChemicalPotential` is a virtual function, it should be implemented in the derived class, i.e. in `FHEngine` in the present case. Same is true for the `getEquilibrium` function.

The segment density fields ϕ and the self consistent field V are instances of the `classScalarField` class. This `ScalarField` class has the operators "`+,-,*,/`" and functions `ln(), exp()` etc.. The operator ∇^2 is implemented as `*p_freePropagator`.

There are several private objects that keep the information on the system. One can get the chain length from the array `d_polymer[]` using the command `d_polymer[i]->numMonomer()`. One can also get the composition from `p_composition`.

```
int FHEngine::getChemicalPotential
    (  const vector<ScalarField*>& phi //  phi ( segment volume fraction )
     , vector<ScalarField*>& fV        //  chemical potential
     , MeanFieldChar* pMeanFieldChar   //  control parameters for the calculation
    )
```

```
{
    FHChar& rFHChar = *( (FHChar*) pMeanFieldChar ); // casting MeanFieldChar onto FHChar.
    int n = p_composition->numSCFUnit();
    int speciesI, speciesJ;
    *fV[ n - 1 ] = 0.;
    double polymerLength = d_polymer[ n - 1 ]->numMonomer();
    ScalarField engN = ( phi[ n - 1 ]->log() ) / polymerLength + 1. / polymerLength;
    for( int i = 0; i < n - 1; i++ ) {
        polymerLength = d_polymer[ i ]->numMonomer();
        // enthalopy part: ln( phi ) / N + 1 / N
        *fV[ i ] =  ( phi[ i ]->log() ) / polymerLength + 1. / polymerLength - engN;
        speciesI = p_composition->stateId( p_composition->SCFUnit( i ) );
        // enthalpy part: chi * phi
        for( int j = 0; j < n; j++ ) {
            speciesJ = p_composition->stateId( p_composition->SCFUnit( j ) );
            *fV[ i ] += *phi[ j ] * p_chi[ speciesI ][ speciesJ ];
        }
        *fV[ i ]     -= *phi[ i ] * p_chi[ speciesI ][ speciesJ ];
        *fV[ n - 1 ] += *fV[ i ];
        *fV[ i ] -= ( *p_freePropagator * *phi[ i ] ) * ( 2. * rFHChar.d_kappas[ i ] );
    }
    *fV[ n - 1 ] *= - 1.;
    *fV[ n - 1 ] -=   ( *p_freePropagator * *phi[ n - 1 ] )
                    * ( 2. * rFHChar.d_kappas[ n - 1 ] );
    return 1;
}
```

As is shown above, the total length of the program code is not so large. The program can be easily written with the use of the objects (private members and arguments). Please refer the header files for `ScalarField, FreePropagator, Polymer`, and `Composition` for the usage of the objects.

The parameters of the physical system is stored in FHChar class, which is a derived class of `classMeanFieldChar`. In the present case, only a variable κ is added as follows.

```
class FHChar :public MeanFieldChar{
public:
    FHChar(): MeanFieldChar( MeanFieldChar::FH );
    ....
    vector< pair< int, double > > d_kappaData; // pair of the polymer ID and kappa.
};
```

This variable can be accessed through a public method of the class SCFEngineDataIO, i.e. the I/O interface class of SUSHI.

```
class SCFEngineDataIO {
    .....
    FHChar*  p_FHChar;      //  Public member.
    .....
    MeanFieldChar* getEngineControlChar()  //  SUSHI calls this function.
    FHChar*        newFHChar( classOfInterface& interfaceObject );
                            // This function is called by the above function.
                            // The argument "classOfInterface" means the I/O interface
                            // objects such as the UDF object or the SEED object.
                            // Off course, the user can modify or replace the interface
                            // as he/she likes.
```

```
    .....
}

MeanFieldChar* SCFEngineDataIO::getEngineControlChar() {
    switch( d_solver ) { //  This function returns the appropriate controller class
                         //  depending on the solver type "d_solver".
    case MeanFieldChar::ADF :
        return getADFChar();
    case MeanFieldChar::FH :
        return getFHChar();
    case MeanFieldChar::SCF :
    default:
        ;
    }
    return getSCFChar();
}

FHChar* SCFEngineDataIO::newFHChar( classOfInterface& interfaceObject ) {
    //  Please refer the source code for the detail.
    ....
}
```

The source codes of the interface class for the "SUSHIInput.udf" format are generated with the use of the "makeinterface" program that is offered by GOURMET. Please refer the GOURMET manual for the detail.

After completing the writing of the program codes, one has to registrate "EngineGenerator" class. The private functions and the data needed for the class "EngineGenerator" are as follows.

```
class EngineGenerator {
....
private:
    void setEngine
        (  const vector<Polymer*>& polymer
         , const vector<double>&   polymerVolumeFraction
         , const vector<Solvent*>& solvent
         , const vector<double>&   solventVolumeFraction
         , const ChiParameter& chiParameter
         , const MFBaseMesh& mesh
         , const PhysicalBoundaryCondition& physicalBc
         , const map<string, MonomerMeanFieldChar* >* pMonomerMeanFieldCharTable
         , const FreePropagator* pFreePropagator
         , const MeanFieldChar* pMeanFieldChar
        );
    ....
    FHEngine*  p_fhEngine;
    ....
};
```

The actual implementation of the `EngineGenerator.setEngine` is as follows.

```
    switch( pMeanFieldChar->type() ) {
    ....
    case MeanFieldChar::FH :
        p_fhEngine = new FHEngine
             (  polymer, polymerVolumeFraction
              , solvent, solventVolumeFraction
```

```
            , chiParameter, mesh, physicalBc
            , pMonomerMeanFieldCharTable
            , pFreePropagator
            , pMeanFieldChar
           );
      p_engine = p_fhEngine;
      break;
  ....
  }
```

The classes for the dynamic simulations can be used commonly. Thus, by writing only a code for the evaluation of the chemical potential, one can obtain a program for the dynamic simulations. The blend2D_4_FH_dy_uin.udf is a sample input file. The result of a simulation with $\kappa = 0.1$ is shown in the following Figure.

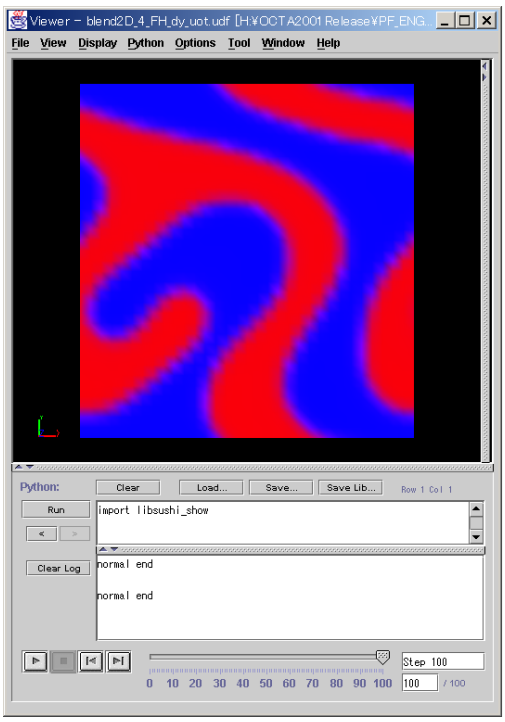


Figure C.1: Dynamics of an A/B polymer blend obtained by FHEngine

Appendix D

Interface Simulator: An example of the system extension for solving interfacial problems

In this appendix, we show, using the interfacial problems as a target, three examples of the extensions of SUSHI described in the previous appendix. Source programs for these extended simulators are stored under the "InterfaceSimulator" directory. The names of these simulators are

- FluidSimulator(fluid)
- MicelleSimulator(micelle)
- SurfaceSimulator(surface).

In these simulators, the input data can be supplied as UDF files. The output of these simulators consists of many text files. The execution of the simulators is started from the command line by entering the following command.

```
fluid -I fluid.udf, etc.
```

Since the simulator will create a large number of output files with fixed names, it is desirable to prepare a separate directory for each run. The "MicelleSimulator" is supplemented by a Python script for analyzing the output files. Several samples of the input UDF files are stored under the "udf" directory. One can understand the procedures performed in the simulators by reading the contents of these sample UDF files.

D.1 FluidSimulator

"FluidSimulator" calculates the interfacial structure and the interfacial tension changing the compositions of the target multi-component system. As the components of the system, one can choose polymers and solvents with any architectures. A change in the composition of the system is specified in the input UDF file by a list of the volume fractions (by specifying the volume fractions in the canonical ensemble) of each component. Although the interfacial tension is calculated by SUSHI using the excess free energy, such a single value of the interfacial tension for a certain composition is not useful in the actual applications. In many cases, we need to know the behavior of the interfacial tension as a function of the composition of the system. For example, when a block copolymer is added to a phase-separating homopolymer blend, it is important to know how the interfacial tension is reduced according to the amount of the added block copolymer.

For the above-mentioned applications, two special techniques are adopted in the "FluidSimulator". The first technique is to prepare the initial condition of a simulation using the final configuration generated by a previous simulation. This technique makes the convergence of the SCF iterations faster if the composition listed in the input UDF file is changing gradually. The second technique is the use of the scheme where the excess free energy can be calculated using the canonical ensemble. Here, we should note the fact that when a macroscopic phase separation is simulated using the canonical ensemble, the sum of the compositions of each phase-separated domain weighted by its size does not coincide with the total composition of the system

specified in the input UDF file. This discrepancy originates from the finite size effect of the simulation box, i.e. the volume occupied by the interfacial region between the bulk phases cannot be neglected compared with the total volume of the system. Therefore, in the canonical ensemble case, the compositions of the reference bulk equilibrium states used in the calculation of the excess free energy does not correspond to the compositions specified in the input data. Instead, one should use the compositions of the bulk phases of the phase-separated system obtained in the simulation. Here, we encounter a problem that the determination of the volume occupied by each phase is ambiguous since the interfaces have a finite thickness (there are no means to determine the interfacial position uniquely). The solution is not trivial, but can be derived from the Gibbs-Duhem relation. That is to use the composition of just one of the coexisting phases as the reference composition to calculate the excess free energy of the whole system including other coexisting macroscopic phases. It is then straightforward that the other coexisting phases does not contribute as an excess free energy and just the interfacial excess can be extracted. The output files of "FluidSimulator" includes the spatial profiles of the volume fraction of each component (`phis_xxxx_xxxx.dat`) corresponding to the compositions listed in the input UDF file, and the list of the value of the interfacial tension as well as the compositions of the two coexisting phases (`energyOfInterface.dat`).

D.2 MicelleSimulator

If a copolymer is dissolved to a selection solvent, a micellar solution will be formed. An isolated micelle can be calculated with SUSHI for a blend of a copolymer and a solvent under the canonical ensemble. As the center of mass of the micelle is fixed in such a calculation, the contribution from the translational degrees of freedom of the micelle is not taken into consideration. Therefore, this method is inadequate to calculate the physical properties of a micellar solution, such as the critical micelle concentration (cmc). "MicelleSimulator" is a simulator for treating such micellar solutions. Using this simulator, the structure and the free energy of micelles with given aggregation number can be calculated. The distribution of the aggregation numbers of the micelles for a given polymer concentration is obtained by analyzing these data using a python script.

Simulation of a micelle with given aggregation number can be performed using a special function of SUSHI. The polymers that form the micelle are assumed to obey the canonical ensemble in order to fix the aggregation number explicitly. In order to make the input file for the SUSHI, it is necessary to convert the aggregation number into the volume fraction of the block copolymer. Such a procedure is automatically done in the "MicelleSimulator". On the other hand, the solvents are assumed to obey the grand canonical ensemble. Thus, their bulk volume fractions are specified under the condition that the sum of the bulk volume fractions of solvents must equal to unity because the block copolymer cannot escape from the system.

In order to form a stable micelle in the calculation, the positions of the free ends of the copolymers that form the core of the micelle are restricted within a region around the center of the micelle. For this purpose, the MASK function of SUSHI is used. If the restricted region is too small, the excess free energy will be estimated larger than its true equilibrium value due to the restrictions on the chain conformations. As long as the restricted region is large enough, however, the value of the excess free energy dose not depend on the size of the restricted region. Due to this reason, the calculation of a micelle with a certain aggregation number requires many simulations changing the size of the MASK region. "MicelleSimulator" performs such a series of simulations automatically using the input data on the MASK conditions and on the tolerance in the judgement of the convergence of the SCF iterations. Using the "MicelleSimulator", one can calculate tens or hundreds of micelles with different aggregation numbers in a single simulation run.

The main outputs of the "MicelleSimulator" are as follows.

```
energyPerMonomer.dat       // Excess free energy per monomer.
getMicelle_xxxx_xxxx.dat   // The relations between the MASK size and the excess free energy.
phis_xxxx_xxxx.dat         // Profiles of the volume fractions.
plot_phis_xxxx_xxxx.plt    // Gnuplot commands that are used in plotting the
                           // data in "phis_xxxx_xxxx.dat".
```

Here, "xxxx" means the aggregation number of the micelle (in case of a mixed micelle composed of two or more copolymers, the aggregation numbers of each component).

The data analysis using the Python script is performed as follows.

In the directory where "energyPerMonomer.dat" exists, enter the following command.

```
python distribution.py <N> <mu>
```

Then, the distribution of the aggregation numbers of the spherical micelles is obtained. Here, <N> is the chain length of the polymer that forms the micelle (if the micelle is composed of a single component). <mu> is the Lagrange multiplier for the constraint on the total volume fraction of the polymers.

When the above-mentioned command is executed, a file named "distribution.dat" is created. Similarly, if one enters the following command,

```
python analysis.py <N> <mu0> <mu1>
```

the following three quantities are calculated as a function of the total volume fraction of the polymers; (1) the volume fraction of the polymers which are not forming micelles (unimer volume fraction) (2) the weight-average aggregation number, and (3) the fraction of the number of polymers that form the micelles to the total number of polymers included in the system. The names of the files to be generated are

```
unimerVsOverall.dat
paveVsOverall.dat
micelleFractionVsOverall.dat,
```

respectively.

D.3 SurfaceSimulator

"SurfaceSimulator" is a simulator for calculating the free energy of a system composed of two parallel solid surfaces separated by a polymer solution, and gives the interaction between these surfaces as a function of the distance between the surfaces. In the current version of SUSHI, only flat surfaces can be treated. The polymer chains can be grafted to the surfaces. For each value of the separation distances registered in the input UDF file, the SCF calculations are performed, and the values of the excess surface free energy are written into the output file named "energyOfSurfaces.dat". The other output files are as follows.

```
phis_xxxx.dat         // Profiles of the volume fractions.
plot_phis_xxxx.plt    // Gnuplot commands that are used in plotting the
                      // data in "phis_xxxx.dat".
```

References

1) Helfand, E. and Wasserman, Z. R.: *Macromolecules*, **9**, 879 (1976).

2) Helfand, E. and Wasserman, Z. R.: *Macromolecules*, **9**, 960 (1978).

3) Helfand, E. and Wasserman, Z. R.: *Macromolecules*, **9**, 994 (1980).

4) Hong, K. M. and Noolandi, J.: *Macromolecules*, **14**, 727 (1981).

5) Evers, A., Scheutjens, J. M. H. M., and Fleer, G. J.: *Macromolecules*, **23**, 5221 (1990).

6) Fraaije, J. G. E. M.: *J. Chem. Phys.*, **99**, 9202 (1993).

7) Fleer, G. J., Cohen Stuart, M. A., Scheutjens, J. M. H. M., Cosgrove, T., and Vincent B. : *Polymers at Interfaces* (Chapman & Hall, London, 1993).

8) Kawakatsu, T.: *Statistical Physics of Polymers: An Introduction* (Springer, Berlin, 2004).

9) Honda, T. and Kawakatsu, T.: *Macromolecules*, **39**, 2340 (2006).

10) Honda, T. and Kawakatsu, T.: "Computer simulations on nano-scale phenomena based on the dynamic density functional theory. Applications of SUSHI in OCTA system" in *Nanostructured Soft Matter: Experiments, Theory and Perspectives*, A. V. Zvelindovsky, ed., Springer-Verlag (2007).

11) Bohbot-Raviv, Y. and Wang, Z.-G.: *Phys. Rev. Lett.*, Vol. 85, p. 3428 (2000).

12) Honda, T. and Kawakatsu, T.: *Macromolecules*, **40**, 1227 (2007).

13) Honda, T: *proceeding of Koubunshi Keisanki Kagaku Kenkyukai*, P-2, Tokyo (2014).

14) Lyakhova, K. S., Zvelindovsky, A. V., and Sevink, G. J. A.: *Macromolecules*, **39**, 3024 (2006).

15) Ly, D. Q., Honda, T., Kawakatsu, T., and Zvelindovsky, A. V.: *Macromolecules*, **40**, 2928 (2007).

16) Ly, D. Q., Honda, T., Kawakatsu, T., and Zvelindovsky, A. V.: *Macromolecules*, **41**, 4501 (2008).

17) Ly, D. Q., Honda, T., Kawakatsu, T., and Zvelindovsky, A. V.: *Soft Matter*, **5**, 4814 (2009).

18) Ly, D. Q., Pinna, M., Honda, T., Kawakatsu, T., and Zvelindovsky, A. V.: *J. Chem. Phys.*, in press (2013).

19) Honda, T. and Kawakatsu, T.: *J. Chem. Phys.*, **129**, 114904 (2008).

20) Doi, M.: *Introduction to Polymer Physics*, chapter 5, (Oxford University Press, London, 1996).

21) de Gennes, P. G.: *Scaling Concepts in Polymer Physics*, chapter XIII, (Cornell University Press, Ithaca, 1979).

22) Broseta, D., Fredrickson, G. H., Helfand, E. and Leibler, L.: *Macromolecules*, **23**, 132 (1990).

23) Anastasiadis, S. H., Gancar, I. and Koberstein, J. T.: *Macromolecules*, **21**, 2980 (1988).

24) Helfand, E. and Tagami, Y.: *J. Chem. Phys.*, **62**, 1327 (1975).

25) van Krevelen, D. W.: *Properties of Polymers*, chapters 7 & 8, (Elsevier, Amsterdam, 1990).

26) Brandrup, J., Immergut, E. H. and Grulke, E. A. eds.: *Polymer Handbook (4th Ed.)*, chapter VII, 675, (John Wiley and Sons, New York, 1999).

27) Mark, J. E. ed.: *Physical Properties of Polymers Handbook*, chapter 19, (AIP Press, Woodbury, 1996).

28) Japan, P. S. ed.: *High Functional Polymer Alloys (in Japanese)*, chapter 3, (Maruzen, 1991).

29) Hoftyzer, P. J. and van Krevelen, D. W.: in van Krevelen, D. W. ed., *Properties of Polymers*, chapter 7, pp. 152 – 155, (Elsevier, Amsterdam,1990).

30) Fedors, R. F.: *Polym. Eng. Sci.*, **14**, 147 (1974).

31) Dunkel, M.: *Z. physik. Chem.*, **A138**, 42 (1928).

32) Aoyagi, T., Honda T., and Doi, M.: *J. Chem. Phys.*, **117**, 8153 (2002).